SYBASE®

New Functionality in Sybase® Adaptive Server™ Enterprise 11.9.2

Adaptive Server™

Document ID: 30602-01-1192-01

August 1998

Copyright Information

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

Locking Changes

2. Locking Schemes in Version 11.9.2

3. Creating and Managing Data-Only-Locked Tables

4. Setting Space Management Properties

5. Locking During Query Processing

6. How Locking Schemes Affect Object Sizes

7. New Locking Commands

Statistics and Query Optimization Changes

8. Statistics Enhancements

9. Managing Table and Index Statistics

10. Query Processing Changes

11. Changes to *showplan* and *dbcc traceon(302)* Output

New and Changed Syntax and Commands

12. New and Changed Transact-SQL Commands

13. New and Changed Functions

14. New and Changed System Procedures

15. *optdiag* Utility

System Administration

16. New and Changed Configuration Parameters

17. Using the *reorg* Command

18. System Administration Changes

Changes to Component Integration Services

19. What's New in Component Integration Services

Performance and Tuning Issues

20. Enhancements to *sp_sysmon*

Appendix

A. System Tables

List of Figures

List of Tables

About This Book

This book discusses the new functionality and changes in Sybase® Adaptive Server™ Enterprise version 11.9.2. It supplements the Adaptive Server 11.5.x documentation.

See Chapter 1, "New Functionality in Version 11.9.2," for information about the features introduced in this version.

Audience

This manual is intended for:

- Sybase System Administrators
- Database designers
- Application developers

How to Use This Book

Chapter 1, "New Functionality in Version 11.9.2," describes the changes in this version of Adaptive Server. It introduces the major features and provides a list of changes that may affect existing applications or require changes in system administration.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- 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 SyBooks™-on-the-Web.
- The Adaptive Server installation documentation for your platform – describes installation and upgrade procedures for all Adaptive Server and related Sybase products.
- The Adaptive Server configuration documentation for your platform – describes configuring a server, creating network

connections, configuring for optional functionality, such as auditing, installing most optional system databases, and performing operating system administration tasks.

- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server release 11.5, the system changes added to support those features, and the changes that may affect your existing applications.
- *Navigating the Documentation for Adaptive Server* – an electronic interface for using Adaptive Server. This online document provides links to the concepts and syntax in the documentation that are relevant to each task.
- *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 and user and system databases, and specifying character conversion, international language, and sort order settings.
- *Adaptive Server Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.
- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Programs* manual for your platform – documents the Adaptive Server utility programs, such as **isql** and **bcp**, which are executed at the operating system level.
- *Security Administration Guide* – explains how to use the security features provided by Adaptive Server to control user access to data. This manual includes information about how to add users to Adaptive Server, administer both system and user-defined roles, grant database access to users, and manage remote Adaptive Servers.

- *Security Features User's Guide* – provides instructions and guidelines for using the security options provided in Adaptive Server from the perspective of the non-administrative user.
- *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.
- *Adaptive Server Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Master Index for Adaptive Server Publications* – combines the indexes of the *Adaptive Server Reference Manual*, *Component Integration Services User's Guide*, *Performance and Tuning Guide*, *Security Administration Guide*, *Security Features User's Guide*, *System Administration Guide*, and *Transact-SQL User's Guide*.

Other Sources of Information

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

- Technical Library CD contains product manuals and technical documents and is included with your software. The DynaText browser (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 Technical Library.
- Technical Library Web site includes the Product Manuals site, which is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition, you'll find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase / Powersoft newsgroups.

  To access the Technical Library Web site, go to support.sybase.com, click the Electronic Support Services tab, and select a link under the Technical Library heading.

Sybase Certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

For the latest information on product certifications and/or the EBF Rollups:

1. Point your Web browser to Technical Documents at the following Web site:

   techinfo.sybase.com

2. In the Browse section, click on the Hot entry.
3. Explore your area of interest: Hot Docs covering various topics, or Hot Links to Technical News, Certification Reports, Partner Certifications, and so on.

If you are a registered Support*Plus* user:

1. Point your Web browser to Technical Documents at the following Web site:

   techinfo.sybase.com

2. In the Browse section, click on the Hot entry.
3. Click on the EBF Rollups entry.

   You can research EBFs using Technical Documents, and you can download EBFs using Electronic Software Distribution (ESD).

4. Follow the instructions associated with the Support*Plus*℠ Online Services entries.

If you are not a registered Support*Plus* user, and you want to become one:

You can register by following the instructions on the Web.

To use Support*Plus*, you need:

- A Web browser that supports the Secure Sockets Layer (SSL), such as Netscape Navigator 1.2 or later
- An active support license
- A named technical support contact
- Your user ID and password

Whether or not you are a registered Support*Plus* user:

You may use Sybase's Technical Documents. Certification Reports are among the features documented at this site.

1. Point your Web browser to Technical Documents at the following Web site:

   techinfo.sybase.com

2. In the Browse section, click on the Hot entry.
3. Click on the topic that interests you.

Conventions

The following section describes the 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 in this manual are as follows:

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**
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*`)`

Table 1: Font and syntax conventions in this manual (continued)

Element	Example
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.

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.

For more information, see "Changing the Default Character Set, Sort Order, or Language" in Chapter 13, "Configuring Character Sets, Sort Orders, and Languages," in the *System Administration Guide*.

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

Some 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.

Introduction

1 New Functionality in Version 11.9.2

This chapter describes the new functionality in version 11.9.2. Topics include:

New Locking Schemes

Adaptive Server version 11.9.2 provides two new locking schemes to improve the concurrency and performance of Adaptive Server:

- Datapages locking
- Datarows locking, also known as row-level locking

Together, these new locking schemes are referred to as data-only locking.

The pre-11.9.2 locking scheme continues to be supported; it is called **allpages locking**, and it is the default locking scheme when you first install or upgrade to version 11.9.2. A System Administrator can specify any locking scheme as the server-wide default.

Users can specify a locking scheme for a newly created table, using the **create table** command, and can change the locking scheme for an existing table to any other locking scheme using the **alter table** command.

Some of the changes made to support these new locking schemes include:

- Additional types of locks
- Changes to table and index structures for tables using the new locking schemes
- New configuration parameters and changes to existing configuration parameters
- Additions and changes to Transact-SQL command syntax, including:
  - Additions to **create table** and **alter table** to allow specifying the locking scheme
  - A new **reorg** command to manage space in tables that use the new locking schemes
  - Changes to **select into** syntax to allow specifying the locking scheme on the created table
- Changes to system procedures to allow reporting on and configuring of new functionality

The following chapters discuss the locking-related changes in this version of Adaptive Server:

Chapter	Topics Covered
Chapter 2, "Locking Schemes in Version 11.9.2"	An overview of locking changes and new types of locks
Chapter 3, "Creating and Managing Data-Only-Locked Tables"	How to specify or change locking schemes and how to determine which locking scheme to choose
Chapter 4, "Setting Space Management Properties"	How to set properties for tables and indexes to improve space usage and performance and reduce the frequency of maintenance operations.
Chapter 5, "Locking During Query Processing"	Types and duration of locks that are held for queries for combinations of isolation levels and locking schemes
Chapter 6, "How Locking Schemes Affect Object Sizes"	Effects of new locking schemes on table and row sizes; effects of index enhancements on index size

Chapter	Topics Covered
Chapter 7, "New Locking Commands"	New commands: • **lock table** explicitly locks an entire table • **set lock wait** specifies how long a command should wait for locks • Additional syntax for **select**, **readtext**, and data modification commands to skip all locked pages or rows • Transaction isolation level 2 provides repeatable reads
Chapter 16, "New and Changed Configuration Parameters"	A guide to the new and changed configuration parameters that affect locking and lock management
Chapter 17, "Using the reorg Command"	How to use the **reorg** command to maintain data-only-locked tables.
Chapter 20, "Enhancements to sp_sysmon"	Additions to **sp_sysmon** output for reporting on data-only-locked tables

➤ ***Note***

The number of locks available to all processes on the server is limited by the configuration parameter **number of locks**. Changing to data-only locking affects the number of locks required during query processing. See "Estimating number of locks for Data-Only-Locked Tables" on page 18-18 for more information.

Changes to Statistics and Query Optimization

Adaptive Server version 11.9.2 increases the flexibility of the statistics used by the query optimizer. Optimizer statistics are now kept in two system tables, *systabstats* and *sysstatistics*. A new utility, **optdiag**, can extract statistics and allows editing of statistics. **optdiag** makes many of these statistics viewable for the first time.

Statistics are now kept on a per-column basis, rather than a per-index basis. Query costing in the optimizer has been enhanced to use column-level statistics for search arguments and joins, even if indexes do not exist on the column.

The **update statistics** command now supports storing statistics for unindexed columns. New **update index statistics** syntax facilitates creating statistics on all columns used in an index, and **update all**

statistics now generates statistics for all columns in a table. A new command, **delete statistics**, can be used to drop column-level statistics, since **drop index** no longer removes statistics.

New statistics are kept to support the new locking schemes. Additional statistics track the clustering of data rows and data pages in physical storage and improve the cost estimates made by the optimizer. Some queries on tables using new locking schemes are optimized differently than queries on tables using the old locking scheme.

The following sections discuss statistics and query processing changes:

Chapter or Section	Topic
Chapter 8, "Statistics Enhancements"	Overview of statistics changes and the output of **optdiag**
Chapter 9, "Managing Table and Index Statistics"	How to use **update statistics** to create statistics on unindexed columns, and how to use the new **delete statistics** command
Chapter 10, "Query Processing Changes"	How new statistics are used in query processing, and how query processing may differ for data-only-locked tables.
Chapter 11, "Changes to showplan and dbcc traceon(302) Output"	Changes to **showplan** and **dbcc traceon(302)** output
Chapter 15, "optdiag Utility"	How to use the **optdiag** utility to display and edit statistics
Appendix A, "sysstatistics" on page A-7 and "systabstats" on page A-8	New system tables that store statistics

Enhancements to the *create index* Command

There are two enhancements to **create index**:

- The **create index** command allows the user to specify ascending or descending order for each column in the index. In earlier versions, all indexes were created in ascending order for all columns. Scans that needed to read the data in reverse order could scan the index backward, but if the required order was a mix of ascending and descending order on the keys, the query needed to perform a sort step. Performance can be improved by

matching the index ordering to the ordering used by most queries.

- Indexes can include as many as 31 key columns, an increase from 16 in previous versions. The maximum total number of bytes allowed for index keys is 600.

For more information, see:

- "Improved Performance for order by Queries" on page 10-21
- **create index** on page 12-14

Changes and Additions to Transact-SQL Syntax

Adaptive Server version 11.9.2 provides new commands and options that affect locking:

- **readpast** option
- **lock table** command
- Server-level and session level options that specify how long tasks wait for locks
- Support for repeatable reads transaction isolation level (level 2) for data-only-locked tables

For syntax and usage information, see Chapter 12, "New and Changed Transact-SQL Commands."

New *readpast* Concurrency Option

select, **delete**, **update, readtext**, and **writetext** commands now include the readpast option. This option allows the command to skip all pages or rows that are not immediately accessible due to incompatible locks. See "Readpast Locking for Queue Processing" on page 7-5.

New *lock table* Command

A new command, **lock table**, explicitly locks entire tables in either shared or exclusive mode. See "Explicitly Locking a Table with the lock table Command" on page 7-1.

Specifying a Wait Time for Locks

In earlier versions of Adaptive Server and SQL Server, all commands that cannot immediately acquire a lock block, or wait, until the required lock is available. In version 11.9.2, the maximum length of time that a task waits can be specified on a server-wide or session-level basis. Processes that fail to acquire the lock within the specified waiting period fail with an error message. See "Session-Level and System-Level Time Limits on Waiting for a Lock" on page 7-3.

- For information on the new configuration parameter, **lock wait period**, see "lock wait period" on page 16-5.
- For information on the new **set** option, **set lock wait**, see "Session-Level and System-Level Time Limits on Waiting for a Lock" on page 7-3.

Repeatable Read Transaction Isolation

Version 11.9.2 adds support for repeatable read transactions (isolation level 2). See "Repeatable Reads Isolation Level" on page 7-9.

Configurable Database Recovery Order

You can now specify the order in which user databases are recovered when you boot the server. This allows your most critical databases to be recovered first. See "User-Defined Database Recovery Order" on page 18-11.

Fault Verification for *dbcc checkstorage* Faults

The **dbcc checkstorage** command, introduced in Adaptive Server version 11.5, checks the consistency of database objects without blocking access by other tasks. Since user activity can produce soft faults, **dbcc checkstorage** attempts to resolve the fault by a second check; if the fault persists, it is recorded in the *dbccdb* database. Since heavy database use during **dbcc checkstorage** operations can lead to lengthy fault reports, the **dbcc checkverify** command introduced in version 11.9.2 rechecks these faults, while holding a lock on the object, in order to eliminate the soft faults from the list of faults. See "Verifying Faults with dbcc checkverify" on page 18-14.

License Use Monitor

System Administrators can determine whether the usage on their Adaptive Server system exceeds license agreements by using the license monitor feature. Once a day, it records the maximum number of users in a system table in the *master* database. See "Using the License Use Monitor" on page 18-19.

Dynamic SQL Performance Improvements

In pre-11.9.2 releases, the simultaneous use of Dynamic SQL by a large number of users performs poorly, largely due to contention on system tables in *tempdb*. In version 11.9.2, the contention problems have been eliminated, and the performance of the **prepare** and **deallocate** statements has been significantly improved. There are no user interface changes. All changes are internal, so applications do not need to be modified.

Direct Updates Through Joins

In version 11.9.2, many restrictions on when direct updates can be performed have been removed. See Chapter 10, "Update Mode and Updates Through Joins."

Component Integration Services Changes

Component Integration Services for Adaptive Server version 11.9.2 adds the following enhancements for remote table access:

- Enhances the performance of queries that include remote tables
- Simplifies the mapping of local proxy tables to remote tables
- Adds new server classes for Adaptive Server Enterprise, Adaptive Server Anywhere™, and Adaptive Server IQ™
- Adds a new trace flag that, when enabled, makes any query that is part of a **declare cursor** command, and that references proxy tables, read-only by default

For more information, see Chapter 19, "What's New in Component Integration Services."

Character Set Changes

Adaptive Server version 11.9.2 includes support for the European currency symbol, or "Euro". See Chapter 18, "European Currency Symbol."

Adaptive Server can now perform character set conversions that cause a change in the data length. See Chapter 18, "Character-Set Conversions That Change Data Lengths."

Changes That May Affect Existing Applications

Changes in 11.9.2 that may affect user applications and system administration are:

- Changes to statistics and query optimization can cause changes to query plans, even for tables where the locking scheme is not changed.
- More logging is performed than in previous versions. See "New Log Space Requirements" on page 18-1.
- For standby or report-processing servers, new syntax is required to dump the transaction log for the production server and bring the database online. It is no longer possible to load a dump that contains open transactions, bring the database online, and then load a subsequent dump of the log. See "Changes to the dump, load, and online Commands" on page 18-6.
- The **update statistics** command may run more slowly, because it now performs a table scan as well as scanning indexes. Concurrency is improved, because **update statistics** with a table name or index name operates at transaction isolation level 0 on data-only-locked tables, and does not acquire table locks on allpages-locked tables.
- The **update all statistics** command now generates statistics for all columns in a table. It performs a complete table scan for each column that is not the leading column in an index. If your system runs scripts that include **update all statistics**, you may want to replace **update all statistics** commands with **update statistics** and **update partition statistics** commands. See "update statistics" on page 12-57.

Effects of Changing to Data-Only-Locking

Changing tables to data-only locking may have these effects:

- The number of locks required for your applications may increase, and you may need to change the **number of locks** configuration parameter. See "Estimating number of locks for Data-Only-Locked Tables" on page 18-18.
- On data-only-locked tables with clustered indexes, the default behavior of disallowing duplicate rows and the **ignore_dup_row** option are not enforced during inserts and updates. This changes the behavior of commands that insert duplicate rows and may change the behavior of **create clustered index** if the table contains duplicate rows. See "create index" on page 12-14.
- On data-only-locked tables, the **sorted_data** option to **create index** can be used only immediately following a bulk copy operation that copies into an existing table. Use of the **sorted_data** option is prohibited once additional page allocation operations have been made for the table.
- Bulk copy into data-only-locked tables requires the version of **bcp** and the bulk copy libraries shipped with Adaptive Server version 11.9.2. Older versions of **bcp** and the bulk-copy libraries can still be used to copy into allpages-locked tables. See the *Release Bulletin* for the required version number.
- When using parallel sort for data-only-locked tables, the number of worker processes must be configured to equal or exceed the number of partitions, even for empty tables. The database option **select into/bulkcopy/pllsort** must also be enabled.

Query Optimization Changes and Forced Query Plans

Changes in query optimization provide greater accuracy than in previous versions in costing some queries, especially those performing nonclustered index scans. In version 11.9.2, you can create statistics on unindexed columns, which can further improve query optimization. You should review any queries that are using forced query plans, especially index or join order forcing, to determine whether forcing is still needed or whether the optimizer can provide an improved plan.

Index Forcing and Data-Only-Locked Tables

The choice of index to use for a query can be forced with **index** clause and an index name, as in this command:

```
select title_id, type, price
    from titles (index title_id_ix)
```

Adaptive Server accepts an index ID in place of the keyword **index** and the index name. The following statement forces the use of a clustered index on an allpages-locked table:

```
select title_id, type, price
    from titles (1)
```

When you convert a table with a clustered index to data-only locking, the index ID of the clustered index changes. If you execute a query that specifies index ID 1, the optimizer still uses the clustered index on the table. You should carefully check all query plans that force index ID 1.

➤ ***Note***

Sybase strongly recommends the use of index names, rather than index IDs, for forcing index selection.

Performance After Loading Pre-11.9 Databases

When pre-11.9 databases are loaded into an 11.9.2 Adaptive Server, the upgrade process inserts rows into the statistics tables, based on available statistics. The upgrade process does not scan the tables and indexes to generate the new types of statistics that can be used by the improved query optimizer. For statistics that are new in this version, the upgrade process inserts computed or magic values.

If you load a pre-11.9 database, run **update statistics** on the tables as soon as possible. If you experience performance problems after loading a pre-11.9 database, run **update statistics** for the tables in the application before you attempt other solutions.

Ordering of Results with Data-Only Locked Tables

Clustered indexes on data-only-locked tables may not return rows in clustered key order if there is no **order by** clause. Bulk copy is not guaranteed to copy out a table in clustered key order.

Queries on unpartitioned allpages-locked tables with clustered indexes return rows in the clustered key order if the query does not include a sort (that is, if there are no clauses such as **order by** or **distinct**). Bulk copy also copies the rows out of allpages-locked tables in clustered key order for both partitioned and unpartitioned tables.

Locking Changes

2 Locking Schemes in Version 11.9.2

This chapter provides an overview of the new locking schemes.

This chapter contains the following sections:

Overview of Locking Schemes

Adaptive Server provides two new locking schemes:

- Datapages locking, which locks only the data pages
- Datarows locking, which locks only the data rows

The datapages and datarows locking schemes share many characteristics, including the fact that they do not acquire locks on index pages, so together they are often referred to as the **data-only locking** scheme.

The pre-11.9.2 locking scheme continues to be supported; it is called **allpages locking**. Allpages locking locks the data pages and index pages affected by queries. It is the default locking scheme in version 11.9.2.

The two new system tables, *systabstats* and *sysstatistics*, use datarows locking. All other system tables use allpages locking. You cannot change the locking scheme of a system table.

Allpages Locking

Allpages locking is the only locking scheme available in Adaptive Server prior to version 11.9.2.

When a query updates a value in a row in an allpages-locked table, the data page is locked with an exclusive lock. Any index pages affected by the update are also locked with exclusive locks. These locks are transactional, meaning that they are held until the end of the transaction.

Figure 2-1 shows the locks acquired on data pages and indexes while a new row is being inserted into an allpages-locked table.

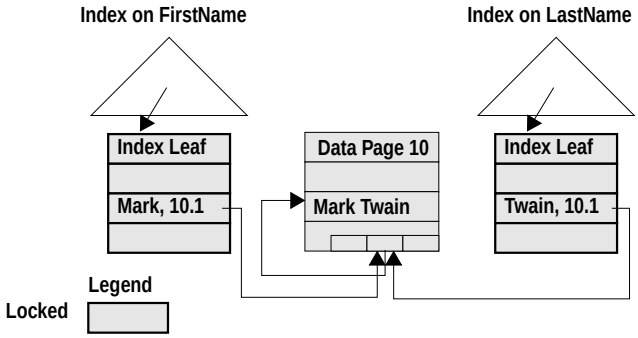

Figure 2-1: Locks held during allpages locking

In many cases, the concurrency problems that result from allpages locking come from the index page locks, rather than the locks on the data pages themselves. Data pages have longer rows than indexes, and may have only 10 to 50 rows per page. If index keys are short, however, an index page can store between 100 and 200 keys. An exclusive lock on an index page can block other users who need to access any of the rows referenced by the index page, a far greater number of rows than on a locked data page.

Datapages Locking

In datapages locking, entire data pages are still locked, but index pages are not locked. When a row needs to be changed on a data page, that page is locked, and the lock is held until the end of the transaction. The updates to the index pages are performed using latches, which are nontransactional. Latches are held only as long as required to perform the physical changes to the page and are then released immediately. Index page entries are implicitly locked by locking the data page. No transactional locks are held on index pages. For more information on latches, see “Latches” on page 2-5.

Figure 2-2 shows an insert into a datapages-locked table. Only the affected data page is locked.

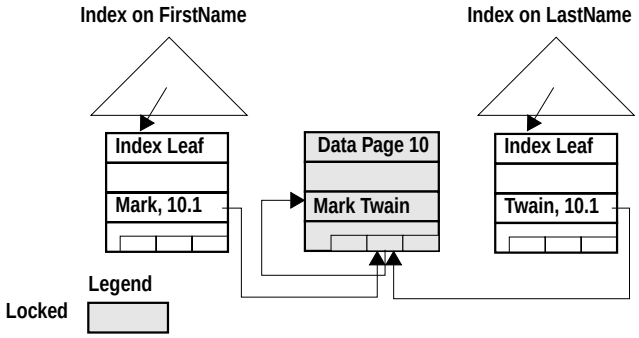

Figure 2-2: Locks held during datapages locking

Datarows Locking

In datarows locking, row-level locks are acquired on individual rows on data pages. Index rows and pages are not locked. When a row needs to be changed on a data page, a nontransactional latch is acquired on the page. The latch is held while the physical change is made to the data page, and then the latch is released. The lock on the data row is held until the end of the transaction. The index rows are updated, using latches on the page, but are not locked. Index entries are implicitly locked by acquiring a lock on the data row.

Figure 2-3 shows an insert into a datarows-locked table. Only the affected data row is locked.

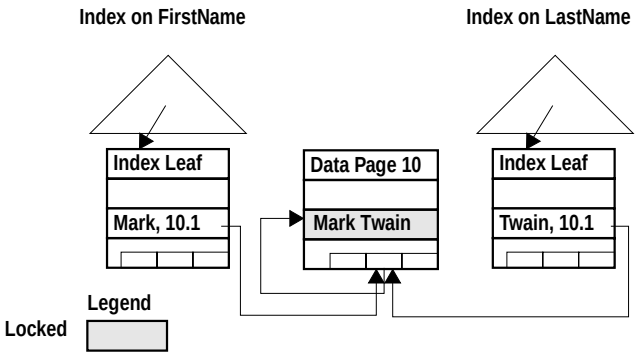

Figure 2-3: Locks held during datarows locking

Similarities Between Allpages Locking and Data-Only Locking

Many principles that apply to page locking for allpages locking also apply to datapages and datarows locking. The use of shared, update, and exclusive locks, the holding of locks during transactions at different isolation levels, and the use of intent and demand locks are major concepts that are used in the same ways. The following describes these behaviors in more detail:

- Data-only locking uses the allpages locking model for intent table locks. Intent locks are held on tables in order to coordinate with shared and exclusive table lock requests. Operations on data-only-locked tables that need shared row or shared page locks acquire shared intent table locks first. Operations that need exclusive row or exclusive page locks acquire exclusive intent table locks first.

  The duration of locks held at different isolation levels and the **holdlock**, **noholdlock**, **serializable**, and **shared** options exhibit the same behavior in all locking schemes, including the use of the **shared** keyword in updatable cursors.
- Row locking also uses demand locking, a queuing mechanism that prevents overlapping shared lock requests from blocking a write transaction for an unacceptably long time. After being skipped by three read transactions, a write transaction is granted a demand lock, and no additional read transactions are allowed to be queued ahead of the write transaction.
- In allpages-locked and datapages-locked tables, the first page of a *text* or *image* column is locked with a shared page lock or an exclusive page lock to prevent access to the page chain. In datarows-locked tables, a shared or exclusive row lock is acquired on row 0 of the first page of the *text* or *image* value. The only difference is in **sp_lock** output.

For more information, see Chapter 5, "Locking During Query Processing," in the *Performance and Tuning Guide*.

New Types of Concurrency Mechanisms

To support data-only locking:

- New row-level lock types have been added
- Page latching provides physical consistency during changes to pages

- New types of locks are used for serializable reads

Row-Level Locks

The row-level locks for datarows-locked tables are similar in function to the page-level locks.

- Shared row locks allow concurrent reads of a row, but they block writes. Shared row locks are applied at all transaction isolation levels higher than level 0 (dirty reads.)
- Update row locks are acquired during the read phase of an update. Update row locks allow other shared row locks, but they block other update locks or exclusive locks. If the row needs to be changed, update row locks are promoted to exclusive row locks, once other shared row locks have been released.
- Exclusive row locks allow a single task to modify a row, but they prevent other tasks from accessing the row until the transaction that acquired the lock completes.

sp_lock reports these locks as “Sh_row”, “Update_row”, and “Ex_row”, respectively.

Latches

Latches are nontransactional synchronization mechanisms used to guarantee the physical consistency of a page. While rows are being inserted, updated or deleted, only one Adaptive Server process can have access to the page at the same time. Otherwise, changes could overwrite each other. Latches are used for both datapages and datarows locking. They are not used for changes to tables using the allpages locking scheme.

The most important distinction between a lock and a latch is the duration:

- A lock can persist for a long period of time: while a page is being scanned, for the duration of a statement, or for the duration of a transaction.
- A latch is held only for the time required to insert or move a few bytes on a data page, to copy pointers, columns or rows, or to acquire a latch on another index page.

Locking for Range Queries at Transaction Isolation Level 3

Rows that can appear or disappear from a results set are called phantoms. Queries that require **phantom** protection (queries at transaction isolation level 3) use a new type of lock, a range lock, for some queries.

Transaction isolation level 3 requires serializable reads within the transaction. A query at transaction isolation level 3 that performs two read operations with the same query clauses should return the same set of results each time. No other task can be allowed to:

- Modify one of the result rows so that it no longer qualifies for the serializable read transaction, by updating or deleting the row
- Modify a row that is not included in the serializable read result set so that the row now qualifies, or insert a row that would qualify for the result set

Adaptive Server uses range locks, infinity key locks, and next-key locks to protect against phantoms on data-only-locked tables. Allpages-locked tables protect against phantoms by holding locks on the index pages for the serializable read transaction.

Figure 2-4 shows two simultaneous transactions. If T1 requires that both **select** queries return the same results, it must be run at

transaction isolation level 3, to prevent the row inserted by T2 from appearing as a phantom.

T1	Event Sequence	T2
`begin transaction`	T1 and T2 start	`begin transaction`
`select * from account where acct_number < 25`	T1 queries a certain set of rows	
	T2 inserts a row that meets the criteria for the query in T1	`insert into account (acct_number, balance) values (19, 500)`
	T2 ends	`commit transaction`
`select * from account where acct_number < 25`	T1 makes the same query and gets a new row–the phantom	
`commit transaction`	T1 ends	

Figure 2-4: A phantom appearing in a range query

Range Locks on Data-Only-Locked Tables

When a query at transaction isolation level 3 (serializable read) performs a range scan using an index, all the keys that satisfy the query clause are locked for the duration of the transaction. Also, the key that immediately follows the range is locked, to prevent new values from being added at the end of the range. If there is no next value in the table, an **infinity key lock** is used as the next key, to ensure that no rows are added after the last key in the table.

Range locks can be shared, update, or exclusive locks; depending on the locking scheme, they are either row locks or page locks. **sp_lock** output shows "Fam dur, Range" in the *context* column for range locks. For infinity key locks, **sp_lock** shows a lock on a nonexistent row, row 0 of the root index page and "Fam dur, Inf key" in the *context* column.

Every transaction that performs an insert or update to a data-only-locked table checks for range locks.

Logical Deletes and Space on Data Pages

In allpages-locked tables, the space used on a data page or an index page is contiguous. When you delete rows or perform updates that change the size of rows, other rows move on the page so that space is contiguously filled from the top of the page downward, as shown in Figure 2-5.

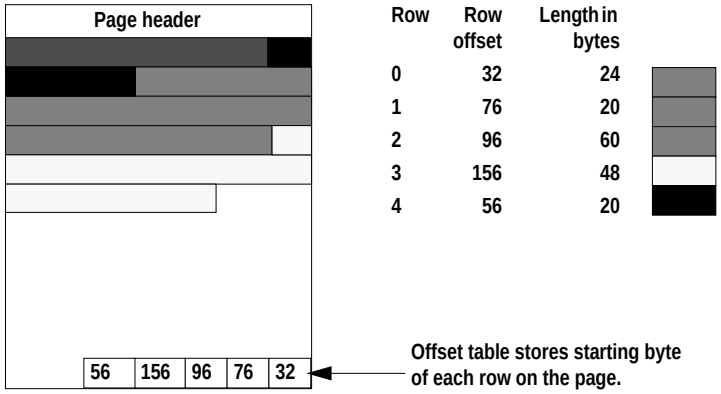

Figure 2-5: Contiguous space on a data page in an allpages-locked table

In an allpages-locked table, deleting row 2 (the row with the cross-hatching in Figure 2-5) would cause the next row to move up on the page immediately, changing the row offset value for row 3.

Deleting rows or reducing the length of rows in data-only-locked tables does not immediately reorganize space on the page to make the data or index entries contiguous. In data-only-locked tables and their indexes, delete operations set a bit for the row to indicate that the row has been logically deleted, but the data or index row is not physically deleted from the page. This ensures that space is available to roll back the change, if needed, but allows other transactions to access rows on the page before the deleting transaction commits.

How Deleted Rows Are Cleared

After a transaction has committed, any of the following actions causes the physical removal of the data row:

- A task that needs to insert a row onto a page or increase the size of a row on the page finds that there is not enough space, but that the page contains rows that were deleted by transactions that are

now committed. If clearing the deleted rows can free enough space for the insert, it clears space used by committed deletes.

- The housekeeper task removes deleted rows from data and index pages, if space reclamation is enabled with the configuration parameter **enable housekeeper GC** and the **housekeeper free write percent** is a nonzero value. See "New Housekeeper Tasks" on page 18-22 for more information.
- The **reorg compact**, **reorg rebuild**, and **reorg reclaim_space** commands reclaim the space left by deleted rows and shrinking updates. See Chapter 17, "Using the reorg Command," for more information.

Fixed Row IDs in Data-Only-Locked Tables

One of the major differences between tables that use data-only locking and those that used allpages locking is that fixed row IDs are used in data-only-locked tables. A row ID is a combination of the page number and the row number; for example, the row ID for row 5 on data page 1137 is (1137,5).

Once a row has been inserted into a data-only-locked table, its row ID never changes as a result of inserts, page splits, or updates that change the row length.

Row IDs for data-only-locked tables change only when clustered indexes are created or re-created and during some **reorg** commands. They never change during data modification statements.

How Row IDs Change in Allpages-Locked Tables

In allpages-locked tables, data modification can cause rows to move to different pages, requiring updates to the row IDs in all of the index entries for the affected rows. Some examples of actions that can cause row IDs to change are as follows:

- When an allpages-locked table has a clustered index:
  - Data modification operations can cause page splits. Page splits occur when a new row needs to be inserted on a full data page or an update increases the length of a row so that it no longer fits on the data page.
  - Updates to key values in the clustered index can cause the row to move to another page, requiring updates to all the nonclustered indexes on the table.

- Some updates are performed on allpages-locked tables by deleting the row from its original location and inserting it on another data page.

Page splitting and updates in an allpages-locked table can be a serious cause of contention, and are one reason for considering a data-only locking scheme for a table. The use of fixed row IDs in data-only locked tables greatly reduces index maintenance costs and concurrency problems caused by blocking on index pages.

Row Forwarding Replaces Page Splits

If an update increases the length of a row in a data-only-locked table so that it no longer fits on the same page:

- The row is inserted onto a different page, and
- A pointer to the row ID on the new page is stored in the original location for the row.

Indexes do not need to be modified when rows are forwarded. All indexes still point to the original row ID for the row.

If the row needs to be forwarded a second time, the original location is updated to point to the new page—the forwarded row is never more than one hop away from its original location. Figure 2-6 shows an update that causes row forwarding.

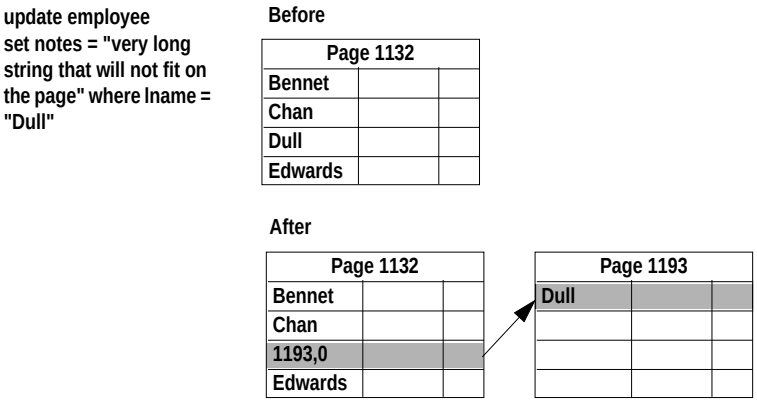

Figure 2-6: Original row stores pointer to the forwarded row's location

Changes to Clustered Indexes for Data-Only-Locked Tables

Page splits on data pages are never performed on data-only-locked tables. This improves concurrency by reducing the number of nonclustered index updates that need to be performed. However, the inability to split data pages is inconsistent with the definition of a clustered index, which requires that data rows be kept in order on the page and that data pages be linked in order.

To provide most of the performance characteristics of clustered indexes for range queries and queries using **order by**, and still provide fixed row IDs, the following adjustments have been made to clustered indexes:

- Clustered indexes on data-only-locked tables use a nonclustered index structure. Indexes created with the **clustered** option have a leaf level above the data level, just like nonclustered indexes. See Chapter 4, "How Indexes Work," in the *Performance and Tuning Guide* for more information on clustered and nonclustered index structure.
- Placement handling for inserts keeps rows physically near each other, according to the clustered index key order. When rows need to be forwarded, allocation algorithms attempt to locate forwarded rows near the original location.

Benefits and Drawbacks of Fixed Row IDs

The major benefit of fixed row IDs is that they can sharply reduce the number of updates to indexes that take place on tables with clustered indexes. With allpages locking, splitting data pages changes the row IDs for approximately half the rows on the page, so the nonclustered index entries for those rows need to be updated with the changed IDs.

The major drawbacks of fixed row IDs are:

- Row forwarding may require extra logical or physical I/O each time the row needs to be read.
- When an insert to a data-only-locked table with a clustered index does not find room on a page to insert a row, it allocates space elsewhere. The clustering of data according to the clustered index order may be affected, leading to increased logical and physical I/O.
- The clustered index on a data-only-locked table has a leaf level above the data level, making the index larger. In some cases, the

index may also be one level deeper, requiring an additional I/O for each indexed access.

Managing Applications That Result in Forwarded Rows

Adaptive Server includes information about the number of forwarded rows in a table in the output of the **optdiag** utility program. See Chapter 15, "optdiag Utility," for more information.

The following tools can help reduce the number of forwarded rows in applications:

- The **create table** command allows you to specify an expected row size. If an application inserts rows that contain many null fields or short fields that are updated later to increase the row size, the application is likely to produce many forwarded rows. Specifying the average size of the final row leaves space on the data pages for the growth of the rows. You can add or change an expected row size for a table with the **sp_chgattribute** system procedure. A server-wide default that is applied to all data-only-locked tables with variable-length columns also helps reduce the number of forwarded rows. See "Reducing Row Forwarding with Expected Row Size" on page 4-1 for more information.
- Adaptive Server's allocation algorithms try to keep rows near a target location. For forwarded rows, the target location is the original row location. The algorithm first looks for room on a page in the same extent as the target and then looks on the same allocation unit as the target. You can increase the likelihood of finding room near the original row location by setting a reserve page gap, specifying a ratio of filled to empty pages with **create table**, **create index**, **alter table**, and **sp_chgattribute**. See "Leaving Space for Forwarded Rows and Inserts" on page 4-8 for more information.
- Re-creating the clustered index on the table copies the table to a new set of data pages, so it eliminates forwarded rows.
- The **reorg forwarded_rows** command removes forwarded rows from a table, and the **reorg rebuild** command copies a table to a new set of data pages, eliminating forwarded rows as the rows are copied.

Displaying Information About Locks

This section describes changes to the following tools that provide information on tasks and locking changes:

- **sp_lock** system procedure
- The **sp_who** system procedure
- Output of **print deadlock information**

Changes to *sp_lock* Output

sp_lock output includes a new column, *row*, that displays the row number for row locks. In addition, **sp_lock** output displays:

- The new lock types:
  - "Sh_row" – shared row locks
  - "Update_row" – update row locks
  - "Ex_row" – exclusive row locks
- Additional information in the *context* column:
  - "Ind pg" indicates locks on index pages (allpages-locked tables only).
  - "Inf key" indicates an infinity key lock, used on data-only-locked tables for some range queries at transaction isolation level 3.
  - "Range" indicates a range lock.

  These new values may appear in combination with "Fam dur" (which replaces "Sync pt duration") and with each other, as applicable.

Changes to *sp_who* Output

Two new status values may be reported by **sp_who**:

- "latch wait" indicates that a task is waiting for a latch.
- "PLC sleep" indicates that a task is waiting to access a user log cache.

Changes to *print deadlock information* Output

The configuration parameter **print deadlock information** prints detailed information about the tables and pages or rows involved in a deadlock and the types of locks. This output has been enhanced to show new types of locks, including row locks, range locks, and next-key locks, and the SQL text.

Row Locking and Lock Promotion

In versions prior to 11.9.2, SQL Server and Adaptive Server limit the overhead of managing large numbers of locks by promoting page locks to table locks, when a process acquires more than a configurable number of page-level locks. Adaptive Server now supports promotion from row-level locks to table-level locks. Lock promotion settings for row-level locks are configured independently of settings for page-level locks.

The configuration parameters that manage page-level lock promotion have been renamed to indicate their function: **page lock promotion HWM**, **page lock promotion LWM**, and **page lock promotion PCT.** Row lock promotion is configured using the **row lock promotion HWM**, **row lock promotion LWM**, and **row lock promotion PCT** parameters. For more information on these parameters, see Chapter 16, "New and Changed Configuration Parameters."

➤ ***Note***

Lock promotion is always two-tiered: from page locks to table locks or from row locks to table locks. Row locks are never promoted to page locks.

3 Creating and Managing Data-Only-Locked Tables

This chapter contains the following sections:

Commands for Specifying Locking Schemes

The locking schemes in Adaptive Server provide you with the flexibility to choose the best locking scheme for each table in your application and to adapt the locking scheme for a table if contention or performance requires a change. The tools for specifying locking schemes are:

- The **sp_configure** system procedure, to specify a server-wide default locking scheme
- The **create table** command, to specify the locking scheme for newly created tables
- The **alter table** command, to change the locking scheme for a table to any other locking scheme
- The **select into** command, to specify the locking scheme for a table created by selecting results from other tables

Specifying a Server-Wide Locking Scheme with *sp_configure*

The **lock scheme** configuration parameter sets the server-wide default locking scheme. The syntax is:

```
sp_configure "lock scheme", 0,
      {allpages | datapages | datarows }
```

This command sets the default lock scheme for the server to datapages:

```
sp_configure "lock scheme", 0, datapages
```

When you first install or upgrade to version 11.9.2, the default locking scheme is allpages.

To see the current default locking scheme, use:

```
sp_configure "lock scheme"
```

```
Parameter Name   Default     Memory Used Config Value Run Value
---------------- ----------- ----------- ------------ -----------
lock scheme      allpages              0 datapages    datapages
```

Specifying a Locking Scheme with *create table*

You can specify the locking scheme for a new table with the **create table** command. The syntax is:

```
create table table_name (column_name_list)
   [ lock {datarows | datapages | allpages } ]
```

If you do not specify the lock scheme for a table, the default value for your server is used, as determined by the setting of the **lock scheme** configuration parameter.

This command specifies datarows locking for the *new_publishers* table:

```
create table new_publishers
(pub_id     char(4)     not null,
 pub_name   varchar(40) null,
 city       varchar(20) null,
 state      char(2)     null)
lock datarows
```

Specifying the locking scheme with **create table** overrides the default server-wide setting. See “Specifying a Server-Wide Locking Scheme with sp_configure” on page 3-1 for information on setting the server-wide locking scheme.

You may also want to specify space management properties for tables when you create them. See Chapter 4, “Setting Space Management Properties,” for more information.

For the complete syntax of **create table**, see page 12-22.

Changing a Locking Scheme with *alter table*

Use the **alter table** command to change the locking scheme for a table. The syntax is:

```
alter table table_name
   lock { allpages | datapages | datarows }
```

This command changes the locking scheme for the *titles* table to datarows locking:

```
alter table titles lock datarows
```

alter table supports changing from one locking scheme to any other locking scheme. You can change:

- From allpages locking to datapages locking and vice versa
- From allpages locking to datarows locking and vice versa
- From datapages locking to datarows locking and vice versa

Changing from allpages locking to data-only locking requires copying the entire table and re-creating any indexes on the table. The operation takes several steps and requires sufficient space to make the copy of the table and indexes. The time required depends on the size of the table and the number of indexes.

Changing from datapages locking to datarows locking or vice versa does not require copying data pages and rebuilding indexes. Switching between data-only locking schemes only updates system tables, and completes in a few seconds.

➤ ***Note***

You cannot use data-only locking for tables that have extremely long rows. For data-only-locked tables with only fixed-length columns, the maximum user data row size is 1958 bytes (or 1960 including the 2 bytes for the offset table). Tables with variable-length columns require 2 additional bytes for each column that is variable-length (this includes columns that allow nulls.) See Chapter 6, “How Locking Schemes Affect Object Sizes,” for more information.

Before and After Changing Locking Schemes

Before you change from allpages locking to data-only locking or vice versa, the following steps are recommended:

- If the table is partitioned, and **update statistics** has not been run since major data modifications to the table, run **update statistics** on the table that you plan to alter. **alter table...lock** performs better with accurate statistics for partitioned tables.
- Perform a database dump.
- Set any space management properties that should be applied to the copy of the table or its rebuilt indexes. See Chapter 4, "Setting Space Management Properties," for more information.
- Determine if there is enough space. See "Space Required to Change the Locking Scheme" on page 3-6.
- If any of the tables in the database are partitioned and require a parallel sort:
  - Use **sp_dboption** to set the database option **select into/bulkcopy/pllsort** to true and run **checkpoint** in the database.
  - Set your configuration for optimum parallel sort performance. See Chapter 15, "Parallel Sorting," in the *Performance and Tuning Guide*.

After *alter table* Completes

- Run **dbcc checktable** on the table and **dbcc checkalloc** on the database to insure database consistency
- Perform a database dump

➤ ***Note***

After changing the locking scheme from allpages locking to data-only locking or vice versa, the use of the **dump transaction** command to back up the transaction log is prohibited. You must first perform a full database dump.

Expense of Switching to or from Allpages Locking

Switching from allpages locking to data-only locking or vice versa is an expensive operation, in terms of I/O cost. The amount of time required depends on the size of the table and the number of indexes that must be re-created. Most of the cost comes from the I/O required to copy the tables and re-create the indexes. Some logging is also required.

The **alter table...lock** command performs the following actions when moving from allpages locking to data-only locking or from data-only locking to allpages locking:

- Copies all rows in the table to new data pages, formatting rows according to the new format. If you are changing to data-only locking, any data rows of less than 10 bytes are padded to 10 bytes during this step. If you are changing to allpages locking from data-only locking, extra padding is stripped from rows of less than 10 bytes.
- Drops and re-creates all indexes on the table.
- Deletes the old set of table pages.
- Updates the system tables to indicate the new locking scheme.
- Updates a counter maintained for the table, to cause the recompilation of query plans

If a clustered index exists on the table, rows are copied in clustered index key order onto the new data pages. If no clustered index exists, the rows are copied in page-chain order for an allpages-locking to data-only-locking conversion.

The entire **alter table...lock** command is performed as a single transaction to ensure recoverability. An exclusive table lock is held on the table for the duration of the transaction.

Switching from datapages locking to datarows locking or vice versa does not require copying pages or re-creating indexes. It only updates the system tables. Setting **sp_dboption "select into/bulkcopy/pllsort"** is not required.

Space Management Properties Applied to the Table

During the copy step, the space management properties for the table are used, as follows:

- If an expected row size value is specified for the table, and the locking scheme is being changed from allpages locking to data-

only-locking, the expected row size is applied to the data rows as they are copied. If no expected row size is set, but there is a **max_rows_per_page** value for the table, an expected row size is computed, and that value is used. Otherwise, the default value specified with the configuration parameter **default exp_row_size percent** is used for each page allocated for the table.

- The **reservepagegap** is applied as extents are allocated to the table.
- If **sp_chgattribute** has been used to save a **fillfactor** value for the table, it is applied to the new data pages as the rows are copied.

Space Management Properties Applied to the Index

When the indexes are rebuilt, space management properties for the indexes are applied, as follows:

- If **sp_chgattribute** has been used to save **fillfactor** values for indexes, these values are applied when the indexes are re-created.
- If **reservepagegap** values are set for indexes, these value are applied when the indexes are re-created.

Space Required to Change the Locking Scheme

Changing from allpages locking to data-only locking, or vice versa, makes a copy of the data pages in a table and re-creates all of the indexes on the table. **alter table** does not attempt to predict how much space is needed to alter the locking scheme. It stops with an error message if it runs out of space on any segment used by the table or its indexes. For large tables, this could occur minutes or even hours after the command starts.

You need free space on the segments used by the table and its indexes, as follows:

- Free space on the table's segment must be at least equal to:
  - The size of the table, plus
  - Approximately 20 percent of the table size, if the table has a clustered index and you are changing from allpages locking to data-only locking
- Free space on the segments used by nonclustered indexes must be at least equal to the size of the indexes.

Clustered indexes for data-only-locked tables have a leaf level above the data pages. If you are altering a table with a clustered index from allpages locking to data-only locking, the resulting clustered index

requires more space. The additional space required depends on the size of the index keys.

Large changes to space management properties can affect the space required. See “Checking Space Requirements for Space Management Properties” on page 3-8.

To determine whether there is space to change to or from allpages locking, you need to know:

- The size of the table and its indexes
- The amount of space available on the segment where the table is stored
- The space management properties set for the table and its indexes

Checking Space Used for Tables and Indexes

To see the size of a table and its indexes, use **sp_spaceused**, as follows:

```
sp_spaceused titles, 1
```

If the table has a clustered index, and you are converting from allpages locking to data-only locking, see “Calculating the Sizes of Indexes for Data-Only-Locked Tables” on page 6-9 for information on estimating the size of the clustered index.

Checking Space on Segments

Tables are always copied to free space on the segment where they are currently stored, and indexes are re-created on the segment where they are currently stored. To determine the number of pages available on the segment where a table is stored, use the system procedure **sp_helpsegment**. The last line of **sp_helpsegment** output prints the total number of free pages available on a segment.

The following command prints segment information for the *default* segment, where objects are stored when no segment was explicitly specified:

```
sp_helpsegment "default"
```

If you do not know the segment name for a table, use **sp_help** and the table name. The segment names for indexes are also reported.

Checking Space Requirements for Space Management Properties

If you make significant changes to space management property values, the table copy can be considerably larger or smaller than the original table. Settings for space management properties are stored in the *sysindexes* tables, and are displayed by **sp_help** and **sp_helpindex**. This output shows the space management properties for the *titles* table:

```
exp_row_size reservepagegap fillfactor max_rows_per_page
------------ -------------- ---------- -----------------
         190             16         90                 0
```

sp_helpindex produces this report:

```
index_name           index_description
    index_keys
    index_max_rows_per_page index_fillfactor index_reservepagegap
    ----------------------- ---------------- --------------------
title_id_ix          nonclustered located on default
    title_id
                          0               75                   0
title_ix             nonclustered located on default
    title
                          0               80                  16
type_price           nonclustered located on default
    type, price
                          0               90                   0
```

Table 3-1 shows how to estimate the effects of setting space management properties.

Table 3-1: Effects of space management properties on space use

Property	Formula	Example
fillfactor	Requires (100 / **fillfactor**) * *num_pages* if pages are currently fully packed	**fillfactor** of 75 requires 1.33 times current number of pages; a table of 1,000 pages grows to 1,333 pages
reservepagegap	Increases space by 1 / **reservepagegap** if extents are currently filled	**reservepagegap** of 10 increase space used by 10%; a table of 1,000 pages grows to 1,100 pages
max_rows_per_page	Converted to **exp_row_size** when converting to data-only-locking	See Table 12-3 on page 12-8.
exp_row_size	Increase depends on number of rows smaller than *exp_rowsize* and the average length of those rows	If **exp_row_size** is 100, and 1,000 rows have a length of 60, the increase in space is: (100 - 60) * 1000 or 40,000 bytes, approximately 20 additional pages

For more information, see “Setting Space Management Properties” on page 4-1.

If There Is Not Enough Space

If there is not enough space to copy the table and re-create all the indexes, determine whether dropping the nonclustered indexes on the table leaves enough room to perform the **alter table...lock** operation on just the table. Without any nonclustered indexes, **alter table...lock** requires space just for the table and the clustered index. You should not drop the clustered index, since **alter table...lock** uses it to order the copied rows, and attempting to re-create it later requires space to make a copy of the table.

After **alter table...lock** completes, re-create the nonclustered indexes.

Sort Performance During *alter table*

If the table being altered is partitioned, parallel sorting is used while rebuilding the indexes. **alter table** performance can be greatly

improved if the data cache and server are configured for optimal parallel sort performance.

During **alter table**, the indexes are re-created one at a time. If your system has enough engines, data cache, and I/O throughput to handle simultaneous **create index** operations, you can reduce the overall time required to change locking schemes by doing the following:

- Drop the nonclustered indexes
- Alter the locking scheme
- Configure for best parallel sort performance
- Re-create two or more nonclustered indexes at once

For information on configuring for parallel sort, see Chapter 15, "Parallel Sorting," in the *Performance and Tuning Guide.*

Specifying a Locking Scheme with *select into*

You can specify a locking scheme when you create a new table, using the **select into** command. The syntax is:

```
select [all | distinct] select_list
       into [[database.]owner.]table_name
       lock {datarows | datapages | allpages }
from ...
```

If you do not specify a locking scheme with **select into**, the new table uses the server-wide default locking scheme, as defined by the configuration parameter **lock scheme**.

This command specifies datarows locking for the table it creates:

```
select title_id, title, price
into bus_titles
lock datarows
from titles
where type = "business"
```

Temporary tables created with the *#tablename* form of naming are single-user tables, so lock contention is not an issue. For temporary tables that can be shared among multiple users, that is, tables created with *tempdb..tablename*, any locking scheme can be used.

Identifying Tables Where Concurrency Is a Problem

If your existing applications experience blocking and deadlock problems, follow the steps below to analyze the problem:

1. Examine deadlocking on the server:
   - Determine whether the cause is application-level deadlocks or internal server-induced deadlocks.
   - Identify the table(s) involved in the deadlock.
2. Check to determine the tables where blocking is a problem.
3. If the table has a clustered index, ensure that performance of the modified clustered index structure on data-only-locked tables will not hurt performance. See "Tables Where Clustered Index Performance Must Remain High" on page 3-15.
4. Convert the locking scheme to datapages locking to determine whether it solves the concurrency problem.
5. Rerun your concurrency tests. If concurrency is still an issue, change the locking scheme to datarows locking.

Examining Deadlocks

Server-side deadlocks are detected and reported to the application by Adaptive Server. In some cases, applications can appear to be deadlocked, but those deadlocks are application-side deadlocks that cannot be detected by the server. Changing to data-only locking cannot solve application-side deadlocks.

Server-Side vs. Application-Side Deadlocks

When tasks deadlock in Adaptive Server, a deadlock detection mechanism finds the deadlock, and rolls back one of the transactions, and sends messages to user and to the Adaptive Server error log. It is possible to induce application-side deadlock situations in which a client opens multiple connections, and these client connections wait for locks held by the other connection of the same application. These are not true server-side deadlocks and cannot be detected by Adaptive Server deadlock detection mechanisms.

Application Deadlock Example

Some developers simulate cursors by using two or more connections from DB-Library™. One connection performs a select and the other

connection performs updates or deletes on the same tables. This can create application deadlocks. For example:

- Connection A holds a shared lock on a page. As long as there are rows pending from Adaptive Server, a shared lock is kept on the current page.
- Connection B requests an exclusive lock on the same pages and then waits.
- The application waits for Connection B to succeed before invoking whatever logic is needed to remove the shared lock. But this never happens.

Since Connection A never requests a lock that is held by Connection B, this is not a server-side deadlock.

Using *print deadlock information*

When you set the configuration parameter **print deadlock information** to 1, Adaptive Server send information about each deadlock to the server's error log. This output identifies the tasks involved in the deadlocks, the types of locks, the commands involved (**select**, **insert**, and so on), the SQL text, and which process was chosen as the victim.

Identifying Tables Where Concurrency Is a Problem

The system procedure **sp_object_stats** prints statistics on lock contention and lock wait times. The syntax is:

```
sp_object_stats interval [, top_n
   [, dbname [, objname [, rpt_option ]]]]
```

To measure lock contention on all tables in all databases, specify only the interval. This command monitors lock contention for 20 minutes, and reports statistics on the 10 tables that had the highest levels of contention:

```
sp_object_stats "00:20:00"
```

Additional arguments to **sp_object_stats** are as follows:

- *top_n* – allows you to specify the number of tables to be included in the report. To report on the top 20 high-contention tables, for example, use:

  ```
  sp_object_stats "00:20:00", 20
  ```

- *dbname* – prints statistics for the specified database
- *objname* – measures contention for the specified table

- *rpt_option* – specifies the report type:
  - **rpt_locks** reports grants, waits, deadlocks and wait times for the tables with the highest contention. **rpt_locks** is the default.
  - **rpt_objlist** reports only the names of the objects that have the highest level of lock activity.

Here is sample output for *titles*, which uses datapages locking:

```
Object Name: pubtune..titles (dbid=7, objid=208003772,lockscheme=Datapages)

 Page Locks     SH_PAGE               UP_PAGE              EX_PAGE$
 ----------    ----------            ----------           ----------
 Grants:            94488                 4052                 4828
 Waits:               532                  500                  776
 Deadlocks:             4                    0                   24
 Wait-time:      20603764 ms           14265708 ms          2831556 ms
 Contention:         0.56%                10.98%               13.79%

*** Consider altering pubtune..titles to Datarows locking.
```

Table 3-2 shows the meaning of the values.

Table 3-2: sp_object_stats output

Output Row	Value
Grants	The number of times the lock was granted immediately.
Waits	The number of times the task needing a lock had to wait.
Deadlocks	The number of deadlocks that occurred.
Contention	The percentage of times that a task had to wait or encountered a deadlock.
Wait-times	The total number of milliseconds that all tasks spent waiting for a lock.

sp_object_stats recommends changing the locking scheme when total contention on a table is more than 15 percent, as follows:

- If the table uses allpages locking, it recommends changing to datapages locking
- If the table uses datapages locking, it recommends changing to datarows locking.

For more information on using **sp_object_stats**, see “sp_object_stats” on page 14-19.

Choosing a Locking Scheme Based on Contention Statistics

If the locking scheme for the table is allpages, the lock statistics reported by **sp_object_stats** include both data page and index lock contention. If lock contention totals 15 percent or more for all shared, update, and exclusive locks, **sp_object_stats** recommends changing to datapages locking. You should make the recommended change, and run **sp_object_stats** again. If contention using datapages locking is more than 15 percent, **sp_object_stats** recommends changing to datarows locking. This two-phase approach is based on these characteristics:

- Changing from allpages locking to either data-only-locking scheme is time consuming and expensive, in terms of I/O cost, but changing between the two data-only-locking schemes is fast and does not require copying the table.
- Datarows locking requires more locks, and consumes more locking overhead. If your applications experience little contention after you convert high-contending tables to use datapages locking, you do not need to incur the locking overhead of datarows locking.

➤ ***Note***

The number of locks available to all processes on the server is limited by the configuration parameter **number of locks**. Changing to datapages locking reduces the number of locks required, since index pages are no longer locked. Changing to datarows locking can increase the number of locks required, since a lock is needed for each row. See “Estimating number of locks for Data-Only-Locked Tables” on page 18-18 for more information.

When examining **sp_object_stats** output, look at tables that are used together in transactions in your applications. Locking on tables that are used together in queries and transactions can affect the locking contention of the other tables. Reducing lock contention on one table could ease lock contention on other tables as well, or it could increase lock contention on another table that was masked by blocking on the first table in the application. For example:

- Lock contention is high for two tables that are updated in transactions involving several tables. Applications first lock *TableA*, then attempt to acquire locks on *TableB*, and block, holding locks on *TableA*. Additional tasks running the same

application block while trying to acquire locks on *TableA*. Both tables show high contention and high wait times.

Changing *TableB* to data-only locking may alleviate the contention on both tables.

- Contention for *TableT* is high, so its locking scheme is changed to a data-only locking scheme. Rerunning **sp_object_stats** now shows contention on *TableX*, which had shown very little lock contention. The contention on *TableX* was masked by the blocking problem on *TableT*.

If your application uses many tables, you may want to convert your set of tables to data-only locking gradually, by changing just those tables with the highest lock contention. Then test the results of these changes by rerunning **sp_object_stats**. You should run your usual performance monitoring tests both before and after the changes are made.

Monitoring and Managing Tables After Conversion

After you have converted one or more tables in an application to a data-only-locking scheme:

- Check query plans and I/O statistics, especially for those queries that use clustered indexes.
- Monitor the tables to learn how changing the locking scheme affects:
  - The cluster ratios, especially for tables with clustered indexes
  - The number of forwarded rows in the table

Applications Not Likely to Benefit from Data-Only Locking

This section describes tables and application types that may get little benefit from converting to data-only locking or may require additional management after the conversion.

Tables Where Clustered Index Performance Must Remain High

If queries with high performance requirements use clustered indexes to return large numbers of rows in index order, you may see some performance degradation if you change these tables to use data-only locking. Clustered indexes on data-only-locked tables are structurally the same as nonclustered indexes. Placement algorithms

keep newly inserted rows close to existing rows with adjacent values, as long as space is available on nearby pages.

Performance for a data-only-locked table with a clustered index should be close to the performance of the same table with allpages locking immediately after a **create clustered index** command or a **reorg rebuild** command, but performance, especially with large I/O, declines if cluster ratios decline because of inserts and forwarded rows.

Performance remains high for tables that do not experience a lot of inserts. On tables that get a lot of inserts, a System Administrator may need to drop and re-create the clustered index or run **reorg rebuild** more frequently. Using space management properties such as **fillfactor**, **exp_row_size**, and **reservepagegap** can help reduce the frequency of maintenance operations. In some cases, keeping the allpages locking scheme for the table, even if there is some contention, may provide better performance for queries performing clustered index scans than using data-only locking for the tables.

Tables with Maximum-Length Rows

Since data-only-locked tables require more overhead per page and per row than allpages-locked tables, the maximum row size for a data-only-locked table is slightly shorter than the maximum row size for an allpages-locked table. For tables with fixed-length columns only, the maximum row size is 1958 bytes of user data for data-only-locked tables. Allpages-locked tables allow a maximum of 1960 bytes.

For tables with variable-length column, subtract 2 bytes for each variable-length columns (this includes all columns that allow null values).

If you try to convert an allpages-locked table that has more than 1958 bytes in fixed-length columns, the command fails as soon as it reads the table schema. When you try to convert an allpages-locked table with variable-length columns, and some rows exceed the maximum size for the data-only-locked table, the **alter table** command fails at the first row that is too long to convert.

4 Setting Space Management Properties

This chapter contains the following sections:

Using Space Management Properties

Earlier versions of Adaptive Server provide **fillfactor** and **max_rows_per_page** to help manage space for tables and indexes. In version 11.9.2, **fillfactor** is supported for indexes on data-only-locked tables, with some additional features for space management. However, **max_rows_per_page** does not provide sufficient flexibility to manage space for data-only-locked tables. Two new space management properties help manage space for data-only-locked tables:

- Expected row size, **exp_row_size**, can be used to reduce row forwarding in data-only-locked tables. If **max_rows_per_page** is set on an allpages-locked table, and the table is converted to data-only locking, the **max_rows_per_page** value is converted to an expected row size value. See "Reducing Row Forwarding with Expected Row Size" on page 4-1.
- Reserve page gap, **reservepagegap**, helps maintain the clustering of data and index leaf pages. See "Leaving Space for Forwarded Rows and Inserts" on page 4-8.
- The **fillfactor** for tables and indexes can be stored using **sp_chgattribute.** The stored value is used by the **reorg rebuild** command and also when the locking scheme of a table is changed from allpages locking to data-only locking or vice versa.

Reducing Row Forwarding with Expected Row Size

Specifying an expected row size for a data-only-locked table is useful when an application allows rows that contain null values or short variable-length character fields to be inserted, and these rows grow in length with subsequent updates.

For example, the *titles* table in the *pubs2* database has many *varchar* columns and columns that allow null values. The maximum row size for this table is 331 bytes, and the average row size (as reported by **optdiag**) is 184 bytes, but it is possible to insert a row with less than 40 bytes, since many columns allow null values. In a data-only-locked table, inserting short rows and then updating them can result in row forwarding. See "Row Forwarding Replaces Page Splits" on page 2-10 for more information.

You can set the expected row size for tables with variable-length columns, as follows:

- With the **exp_row_size** parameter, in a **create table** statement
- With **sp_chgattribute**, for an existing table
- With a server-wide default value, using the configuration parameter **default exp_row_size percent**. This value is applied to all tables with variable-length columns, unless **create table** or **sp_chgattribute** is used to set a row size explicitly or to indicate that rows should be fully packed on data pages.

If you specify an expected row size value for an allpages-locked table, the value is stored in *sysindexes*, but the value is not applied during inserts and updates. If the table is later converted to data-only locking, the **exp_row_size** is applied during the conversion process, and to all subsequent inserts and updates.

Default, Minimum, and Maximum Values for *exp_row_size*

Table 4-1 shows the minimum and maximum values for expected row size and the meaning of the special values, 0 and 1.

Table 4-1: Valid values for expected row size

exp_row_size Values	Minimum, Maximum, and Special Values
Minimum	The greater of: • 2 bytes • The sum of all fixed-length columns
Maximum	Maximum data row length
0	Use server-wide default value
1	Fully pack all pages; do not reserve room for expanding rows

You cannot specify an expected row size for tables that have fixed-length columns only. Columns that accept null values are by definition variable-length columns, since they are zero-length when null.

Default Value

If you do not specify an expected row size or specify a value of 0 when you create a data-only-locked table with variable-length columns, Adaptive Server uses the amount of space specified by the configuration parameter **default exp_row_size percent** for any table that has variable-length columns. See "Setting a Default Expected Row Size Server-Wide" on page 4-5 for information on how this parameter affects space on data pages.

Minimum Value and Maximum Value

The minimum value for expected row size is the greater of:

- The minimum row size, 2 bytes
- The size of all the fixed-length columns for the table

The maximum value is the maximum data row length for the table.

Use **sp_help** to see the defined length of the columns in the table.

Specifying Fully Packed Pages

If you do not want space on a page to be reserved for expanding updates, you can specify a value of 1 for the expected row size, which creates fully packed pages.

Specifying an Expected Row Size with *create table*

This **create table** statement specifies an expected row size of 200:

```
create table new_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit        )
lock datapages
with exp_row_size = 200
```

Adding or Changing an Expected Row Size with *sp_chgattribute*

To add or change the expected row size for a table, use the system procedure **sp_chgattribute**. This sets the expected row size to 190 for the *new_titles* table:

```
sp_chgattribute new_titles, "exp_row_size", 190
```

If you want a table to start using the **default exp_row_size percent** instead of a current, explicit value, use the command:

```
sp_chgattribute new_titles, "exp_row_size", 0
```

To fully pack the pages, rather than saving space for expanding rows, set the value to 1.

Changing the expected row size with **sp_chgattribute** does not immediately affect the storage of existing data. The new value is applied:

- When a clustered index on the table is created or **reorg rebuild** is run on the table. The expected row size is applied as rows are copied to new data pages. If you increase **exp_row_size**, and re-create the clustered index or run **reorg rebuild**, the new copy of the table may require more storage space.
- The next time a page is affected by data modifications.

Setting a Default Expected Row Size Server-Wide

The configuration parameter **default exp_row_size percent** reserves a percentage of the page size to set aside for expanding updates. The default value, 5 percent, sets aside 5 percent of the space available per data page for all data-only-locked tables that include variable-length columns. Since there are 2002 bytes available on data pages in data-only-locked tables, the default value for **default exp_row_size percent** sets aside 100 bytes for row expansion. This command sets the default value to 10 percent:

```
sp_configure "default exp_row_size percent", 10
```

Setting **default exp_row_size percent** to 0 means that no space is reserved for expanding updates for any tables where the expected row size is not explicitly set with **create table** or **sp_chgattribute**.

If an expected row size for a table is specified with **create table** or **sp_chgattribute**, that value takes precedence over the server-wide setting.

Displaying the Expected Row Size for a Table

Use **sp_help** to display the expected row size for a table:

```
sp_help titles
```

If the value is 0, and the table has nullable or variable-length columns, use **sp_configure** to display the server-wide default value:

```
sp_configure "default exp_row_size percent"
```

This query displays the value of the *exp_rowsize* column for all user tables in a database:

```
select object_name(id), exp_rowsize
from sysindexes
where id > 100 and (indid = 0 or indid = 1)
```

Choosing an Expected Row Size for a Table

Setting an expected row size helps reduce the number of forwarded rows only if the rows expand after they are first inserted into the table. Setting the expected row size correctly means that:

- Your application results in a small percentage of forwarded rows
- You do not waste too much space on data pages due to over-configuring the expected row size value

Using *optdiag* to Check for Forwarded Rows

For tables that already contain data, use **optdiag** to display statistics for the table. The "Data row size" shows the average data row length, including the row overhead. This sample **optdiag** output for the *titles* table shows 12 forwarded rows and an average data row size of 184 bytes:

```
Statistics for table:                "titles"

     Data page count:                655
     Empty data page count:          5
     Data row count:                 4959.000000000
     Forwarded row count:            12.000000000
     Deleted row count:              84.000000000
     Data page CR count:             0.000000000
     OAM + allocation page count:    6
     Pages in allocation extent:     1
     Data row size:                  184.000000000
```

You can also use **optdiag** to check the number of forwarded rows for a table to determine whether your setting for **exp_row_size** is reducing the number of forwarded rows generated by your applications. For more information on **optdiag**, see Chapter 8, "Statistics Enhancements."

Querying *systabstats* to Check for Forwarded Rows

You can check the *forwrowcnt* column in the *systabstats* table to see the number of forwarded rows for a table. This query checks the number of forwarded rows for all user tables (those with object IDs greater than 100):

```
select object_name(id) , forwrowcnt
from systabstats
where id > 100 and (indid = 0 or indid = 1)
```

➤ ***Note***

Forwarded row counts are updated in memory, and the housekeeper periodically flushes them to disk. If you need to query the *systabstats* table using SQL, use **sp_flushstats** first to ensure that the most recent statistics are available. **optdiag** flushes statistics to disk before displaying values.

Conversion of *max_rows_per_page* to *exp_row_size*

If a **max_rows_per_page** value is set for an allpages-locked table, the value is used to compute an expected row size during the **alter table...lock** command. The formula is shown in Table 4-2.

Table 4-2: Conversion of max_rows_per_page to exp_row_size

Value of *max_rows_per_page*	Value of *exp_row_size*
0	Percentage value set by **default exp_row_size percent**
255	1 (fully packed pages)
1-254	The smaller of: • Maximum row size • 2002 / **max_rows_per_page** value

For example, if **max_rows_per_page** is set to 10 for an allpages-locked table with a maximum defined row size of 300 bytes, the **exp_row_size** value will be 200 (2002 / 10) after the table is altered to use data-only locking. If **max_rows_per_page** is set to 10, but the maximum defined row size is only 150, the expected row size value will be set to 150.

Monitoring and Managing Tables That Use Expected Row Size

After setting an expected row size for a table, use **optdiag** or queries on *systabstats* to check the number of forwarded rows still being generated by your applications. Run **reorg forwarded_rows** if you feel that the number of forwarded rows is high enough to affect application performance. The **reorg forwarded_rows** command uses short transactions and is very nonintrusive, so it does not cause performance problems if it is run while applications are active. See “Moving Forwarded Rows to Home Pages” on page 17-4 for more information.

If the application still results in a large number of forwarded rows, you may want to increase the expected row size for the table, using **sp_chgattribute**.

You may want to allow a certain percentage of forwarded rows. If running **reorg** to clear forwarded rows does not cause concurrency problems for your applications, or if you can run **reorg** at non-peak times, allowing a small percentage of forwarded rows does not cause a serious performance problem.

Setting the expected row size for a table increases the amount of storage space and the number of I/Os needed to read a set of rows. If the increase in the number of I/Os due to increased storage space is high, then allowing rows to be forwarded and occasionally running **reorg** may have less overall performance impact.

Leaving Space for Forwarded Rows and Inserts

Setting a **reservepagegap** value can reduce the frequency of maintenance activities such as running **reorg rebuild** and re-creating indexes for some tables to maintain high performance. Good performance on data-only-locked tables requires good data clustering on the pages, extents, and allocation units used by the table.

The clustering of data and index pages in physical storage stays high as long as there is space nearby for storing forwarded rows and rows that are inserted in index key order. The **reservepagegap** space management property is used to reserve empty pages for expansion when additional pages need to be allocated.

Row and page cluster ratios are usually 1.0, or very close to 1.0, immediately after a clustered index is created on a table or immediately after **reorg rebuild** is run. However, future data modifications can cause row forwarding and can require allocation of additional data and index pages to store inserted rows. Setting a reserve page gap can reduce storage fragmentation and reduce the frequency with which you need to re-create indexes or run **reorg rebuild** on the table.

Extent Allocation Operations and *reservepagegap*

Commands that allocate many data pages perform **extent allocation** to allocate eight pages at a time, rather than allocating just one page at a time. Extent allocation reduces logging, since it writes one log record instead of eight.

Commands that perform extent allocation are: **select into**, **create index**, **reorg rebuild**, **bcp**, **alter table...lock**, and the **alter table...unique** and **primary key** constraint options, since these constraints create indexes. These commands allocate an extent, and, unless a reserve page gap value is in effect, fill all eight pages.

You specify the **reservepagegap** in pages, indicating a ratio of empty pages to filled pages. For example, if you specify a **reservepagegap** of 8,

an operation doing extent allocation fills 7 pages and leaves the eighth page empty.

These empty pages can be used to store forwarded rows and for maintaining the clustering of data rows in index key order, for data-only-locked tables with clustered indexes.

Since extent allocation operations must allocate entire extents, they do not use the first page on each allocation unit, because it stores the allocation page. For example, if you create a clustered index on a large table and do not specify a reserve page gap, each allocation unit has 7 empty, unallocated pages, 248 used pages, and the allocation page. These 7 pages can be used for row forwarding and inserts to the table, which helps keep forwarded rows and inserts with clustered indexes on the same allocation unit. Using **reservepagegap** leaves additional empty pages on each allocation unit.

Figure 4-1 shows how an allocation unit might look after a clustered index is created with a **reservepagegap** value of 16 on the table. The pages that share the first extent with the allocation unit are not used and are not allocated to the table. Pages 279, 295, and 311 are the unused pages on extents that are allocated to the table.

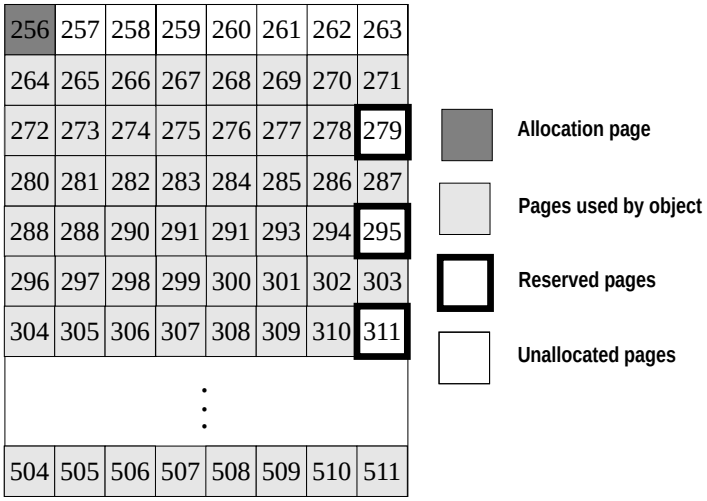

Figure 4-1: Reserved pages after creating a clustered index

Specifying a Reserve Page Gap with *create table*

This **create table** command specifies a **reservepagegap** value of 16:

```
create table more_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit
    )
lock datarows
with reservepagegap = 16
```

A bulk copy operation that copies data into the *more_titles* table leaves 1 empty page for each 15 filled pages.

The default value for **reservepagegap** is 0, meaning that no space is reserved. The maximum value is 255, meaning that 1 page is left unused on each allocation unit.

Specifying a Reserve Page Gap with *create index*

This command specifies a **reservepagegap** of 10 for the nonclustered index pages:

```
create index type_price_ix
on more_titles(type, price)
with reservepagegap = 10
```

You can also specify a **reservepagegap** value with the **alter table...constraint** options, **primary key** and **unique**, that create indexes. This example creates a unique constraint:

```
alter table more_titles
add constraint uniq_id unique (title_id)
with reservepagegap = 20
```

Changing *reservepagegap* with *sp_chgattribute*

The following command uses **sp_chgattribute** to change the reserve page gap for the *titles* table to 20:

```
sp_chgattribute more_titles, "reservepagegap", 20
```

This command sets the reserve page gap for the index *title_ix* to 10:

```
sp_chgattribute "titles.title_ix",
    "reservepagegap", 10
```

The **sp_chgattribute** command only changes values in system tables; data is not moved on data pages as a result of running the procedure. Changing **reservepagegap** for a table affects future storage as follows:

- When data is bulk-copied into the table, the reserve page gap is applied to all newly allocated space, but the storage of existing pages is not affected
- When the **reorg rebuild** command is run on the table, the reserve page gap is applied as the table is copied to new data pages
- When a clustered index is created, the reserve page gap value stored for the table is applied to the data pages.

The reserve page gap is applied to index pages:

- During **alter table...lock**, while rebuilding indexes for the table
- During **reorg rebuild** commands that affect indexes
- During **create clustered index** and **alter table** commands that create a clustered index, as nonclustered indexes are rebuilt

reservepagegap Examples

These examples show how **reservepagegap** is applied during **alter table** and **reorg rebuild** commands.

reservepagegap Specified Only for the Table

The following commands specify a **reservepagegap** for the table, but do not specify a value in the **create index** commands:

```
sp_chgattribute titles, "reservepagegap", 16
create clustered index title_ix on titles(title_id)
create index type_price on titles(type, price)
```

Table 4-3 shows the values applied when running **reorg rebuild** or dropping and creating a clustered index.

Table 4-3: reservepagegap values applied with table-level saved value

Command	Allpages-Locked Table	Data-Only-Locked Table
create clustered index or clustered index rebuild due to **reorg rebuild**	Data and index pages: 16	Data pages: 16 Index pages: 0 (filled extents)
Nonclustered index rebuild	Index pages: 0 (filled extents)	Index pages: 0 (filled extents)

The **reservepagegap** for the table is applied to both the data and index pages for an allpages-locked table with a clustered index. For a data-only-locked table, the table's **reservepagegap** is applied to the data pages, but not to the clustered index pages.

reservepagegap Specified for a Clustered Index

These commands specify different **reservepagegap** values for the table and the clustered index, and a value for the nonclustered *type_price* index:

```
sp_chgattribute titles, "reservepagegap", 16

create clustered index title_ix on titles(title)
    with reservepagegap = 20

create index type_price on titles(type, price)
    with reservepagegap = 24
```

Table 4-4 shows the effects of this sequence of commands.

Table 4-4: reservepagegap values applied with for index pages

Command	Allpages-Locked Table	Data-Only-Locked Table
create clustered index or clustered index rebuild due to **reorg rebuild**	Data and index pages: 20	Data pages: 16 Index pages: 20
Nonclustered index rebuilds	Index pages: 24	Index pages: 24

For allpages-locked tables, the **reservepagegap** specified with **create clustered index** applies to both data and index pages. For data-only-locked tables, the **reservepagegap** specified with **create clustered index** applies only to the index pages. If there is a stored **reservepagegap** value for the table, that value is applied to the data pages.

Choosing a Value for *reservepagegap*

Choosing a value for **reservepagegap** depends on:

- Whether the table has a clustered index,
- The rate of inserts to the table,
- The number of forwarded rows that occur in the table, and
- How often you re-create the clustered index or run the **reorg rebuild** command.

When **reservepagegap** is configured correctly, enough pages are left for allocation of new pages to tables and indexes so that the cluster ratios for the table, clustered index, and nonclustered leaf-level pages remain high during the intervals between regular index maintenance tasks.

Monitoring *reservepagegap* Settings

You can use **optdiag** to check the cluster ratio and the number of forwarded rows in tables. Declines in cluster ratios can also indicate that running **reorg** commands could improve performance:

- If the data page cluster ratio for a clustered index is low, run **reorg rebuild** or drop and re-create the clustered index.
- If the index page cluster ratio is low, drop and re-create the non-clustered index.

If you want to reduce the frequency with which you run **reorg** commands to maintain cluster ratios, increase the **reservepagegap** slightly before running **reorg rebuild**. See Chapter 8, “Statistics Enhancements,” for more information on **optdiag**.

reservepagegap and *sorted_data* Options to *create index*

When you create a clustered index on a table that is already stored on the data pages in index key order, the **sorted_data** option suppresses the step of copying the data pages in key order for unpartitioned

tables. The **reservepagegap** option can be specified in **create clustered index** commands, to leave empty pages on the extents used by the table, leaving room for later expansion. The purposes of these two options are somewhat antithetical, and special rules determine which option takes effect. You cannot use **sp_chgattribute** to change the **reservepagegap** value and get the benefits of both options.

If you specify both options with **create clustered index**:

- On unpartitioned, allpages-locked tables, if the **reservepagegap** value specified with **create clustered index** matches the values already stored in *sysindexes*, the **sorted_data** option takes precedence. Data pages are not copied, so the **reservepagegap** is not applied. If the **reservepagegap** value specified in the **create clustered index** command is different from the values stored in *sysindexes*, the data pages are copied, and the **reservepagegap** value specified in the command is applied to the copied pages.
- On data-only-locked tables, the **reservepagegap** value specified with c**reate clustered index** applies only to the index pages. Data pages are not copied.

Background on the *sorted_data* Option

Besides **reservepagegap**, other options to **create clustered index** may require a sort, which causes the **sorted_data** option to be ignored. For more information, see “Creating an Index on Sorted Data” on page 23-4 of the *Performance and Tuning Guide*. In particular, the following comments relate to the use of **reservepagegap**:

- On partitioned tables, any **create clustered index** command that requires copying data pages performs a parallel sort and then copies the data pages in sorted order, applying the **reservepagegap** values as the pages are copied to new extents.
- Whenever the **sorted_data** option is not superseded by other **create clustered index** options, the table is scanned to determine whether the data is stored in key order. The index is built during the scan, without a sort being performed.

Table 4-5 shows how these rules apply.

Table 4-5: reservepagegap and sorted_data options

	Partitioned Table	Unpartitioned Table
Allpages-Locked Table		
create index with sorted_data and matching **reservepagegap** value	Does not copy data pages; builds the index as pages are scanned.	Does not copy data pages; builds the index as pages are scanned.
create index with sorted_data and different **reservepagegap** value	Performs parallel sort, applying **reservepagegap** as pages are stored in new locations in sorted order.	Copies data pages, applying **reservepagegap** and building the index as pages are copied; no sort is performed.
Data-Only-Locked Table		
create index with sorted_data and any **reservepagegap** value	**reservepagegap** applies to index pages only; does not copy data pages.	**reservepagegap** applies to index pages only; does not copy data pages.

Matching Options and Goals

If you want to redistribute the data pages of a table, leaving room for later expansion:

- For allpages-locked tables, drop and re-create the clustered index without using the **sorted_data** option. Specify the desired **reservepagegap** value in the **create clustered index** command, if the value stored in *sysindexes* is not the value you want.
- For data-only-locked tables, use **sp_chgattribute** to set the **reservepagegap** for the table to the desired value and then drop and re-create the clustered index, without using the **sorted_data** option. The **reservepagegap** stored for the table applies to the data pages. If **reservepagegap** is specified in the **create clustered index** command, it applies only to the index pages.

If you want to create a clustered index without copying data pages:

- For allpages-locked tables, use the **sorted_data** option, but do not specify a **reservepagegap** with the **create clustered index** command. Alternatively, you can specify a value that matches the value stored in *sysindexes*.
- For data-only-locked tables, use the **sorted_data** option. If a **reservepagegap** value is specified in the **create clustered index**

command, it applies only to the index pages and does not cause data page copying.

If you plan to use the **sorted_data** option following a bulk copy operation, a **select into** command, or another command that uses extent allocation, set the **reservepagegap** value that you want for the data pages before copying the data or specify it in the **select into** command. Once the data pages have been allocated and filled, the following command applies **reservepagegap** to the index pages only, since the data pages do not need to be copied:

```
create clustered index title_ix
on titles(title_id)
with sorted_data, reservepagegap = 32
```

Setting *fillfactor* Values with *sp_chgattribute*

When you issue a **create index** command, the **fillfactor** value specified as part of the command is applied as follows:

- Clustered index:
  - On an allpages-locked table, the **fillfactor** is applied to the data pages.
  - On a data-only-locked table, the **fillfactor** is applied to the leaf pages of the index, and the data pages are fully packed (unless **sp_chgattribute** has been used to store a **fillfactor** for the table).
- Nonclustered index – the **fillfactor** value is applied to the leaf pages of the index.

fillfactor values specified with **create index** are applied at the time the index is created. They are not saved in *sysindexes*, and the fullness of the data or index pages is not maintained over time.

When Stored *fillfactor* Values Are Applied

sp_chgattribute allows you to store a **fillfactor** percentage for each index and for the table. The **fillfactor** you set with **sp_chgattribute** is applied:

- When you run **reorg rebuild** to restore the cluster ratios of tables and indexes
- When you use **alter table...lock** to change the locking scheme for a table
- When you run **create clustered index** and there is a value stored for the table

When Stored *fillfactor* Values Are Not Applied

The stored **fillfactor** is not applied when nonclustered indexes are rebuilt as a result of a **create clustered index** command. Creating a clustered index follows the behavior of pre-11.9.2 releases:

- If a **fillfactor** value is specified with **create clustered index**, that value is applied to each nonclustered index.
- If no **fillfactor** value is specified with **create clustered index**, the server-wide default value (set with the **default fill factor percent** configuration parameter) is applied to all indexes.

➤ ***Note***

During **alter table...lock commands**, the **fillfactor** value is applied only when changing from allpages locking to data-only locking, or vice versa. It is not applied for changes between the data-only locking schemes.

fillfactor Examples

No Stored *fillfactor* Values

With no **fillfactor** values stored in *sysindexes*, the **fillfactor** specified in commands that create indexes are applied as shown in Table 4-6.

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

Table 4-6: fillfactor values applied with no table-level saved value

Command	Allpages-Locked Table	Data-Only-Locked Table
create clustered index	Data pages: 80	Data pages: fully packed Leaf pages: 80
Nonclustered index rebuilds	Leaf pages: 80	Leaf pages: 80

The nonclustered indexes use the **fillfactor** specified in the **create clustered index** command. If no **fillfactor** is specified in the **create clustered index** command, the nonclustered indexes always use the server-wide default; they never use a value from *sysindexes*.

Values Used for alter table...lock and reorg rebuild

When no **fillfactor** values are stored, both **alter table...lock** and **reorg rebuild** apply the server-wide default value, set by the configuration parameter **default fill factor percentage**. The default **fillfactor** is applied as shown in Table 4-7.

Table 4-7: fillfactor values applied with during rebuilds

Command	Allpages-Locked Table	Data-Only-Locked Table
Clustered index rebuild	Data pages: default value	Data pages: fully packed Leaf pages: default value
Nonclustered index rebuilds	Leaf pages: default	Leaf pages: default

Table-Level or Clustered Index *fillfactor* Value Stored

This command stores a **fillfactor** value of 50 for the table:

```
sp_chgattribute titles, "fillfactor", 50
```

With 50 as the stored table-level value for **fillfactor,** the following create **clustered index command** applies the **fillfactor** values shown in Table 4-8.

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

Table 4-8: Using stored fillfactor values for clustered indexes

Command	Allpages-Locked Table	Data-Only-Locked Table
create clustered index	Data pages: 80	Data pages: 50 Leaf pages: 80
Nonclustered index rebuilds	Leaf pages: 80	Leaf pages: 80

➤ ***Note***

When a **create clustered index** command is run, any table-level **fillfactor** value stored in *sysindexes* is reset to 0. To affect the filling of data-only-locked data pages during a **create clustered index** or **reorg** command, you must issue **sp_chgattribute** before running the command.

Effects of alter table...lock When Values Are Stored

Stored values for **fillfactor** are used when an **alter table...lock** command copies tables and rebuilds indexes.

Tables with Clustered Indexes

In an allpages-locked table, the table and the clustered index share the *sysindexes* row, so only one value for **fillfactor** can be stored and used for the table and clustered index. You can set the **fillfactor** value for the data pages by providing either the table name or the clustered index name. This command saves the value 50:

```
sp_chgattribute titles, "fillfactor", 50
```

This command saves the value 80, overwriting the value of 50 set by the previous command:

```
sp_chgattribute "titles.clust_ix", "fillfactor", 80
```

If you alter the titles table to use data-only locking after issuing the **sp_chgattribute** commands above, the stored value **fillfactor** of 80 is used for both the data pages and the leaf pages of the clustered index.

In a data-only-locked table, information about the clustered index is stored in a separate row in *sysindexes*. The **fillfactor** value you specify for the table applies to the data pages and the **fillfactor** value you specify for the clustered index applies to the leaf level of the clustered index. When a data-only-locked table is altered to use allpages locking, the **fillfactor** stored for the table is used for the data pages. The **fillfactor** stored for the clustered index is ignored.

Table 4-9 shows the fillfactors used on data and index pages by an **alter table...lock** command, executed after the **sp_chgattribute** commands above have been run.

Table 4-9: Effects of stored fillfactor values during alter table

alter table...lock	No Clustered Index	Clustered Index
From allpages locking to data-only locking	Data pages: 80	Data pages: 80 Leaf pages: 80
From data-only locking to allpages locking	Data pages: 80	Data pages: 80

➤ ***Note***

alter table...lock sets all stored **fillfactor** values for a table to 0.

fillfactor Values Stored for Nonclustered Indexes

Each nonclustered index is represented by a separate *sysindexes* row. These commands store different values for two nonclustered indexes:

```
sp_chgattribute "titles.ncl_ix", "fillfactor", 90
sp_chgattribute "titles.pubid_ix", "fillfactor", 75
```

Table 4-10 shows the effects of a **reorg rebuild** command on a data-only-locked table when the **sp_chgattribute** commands above are used to store **fillfactor** values.

Table 4-10: Effect of stored fillfactor values during reorg rebuild

reorg rebuild	No Clustered Index	Clustered Index	Nonclustered Indexes
Data-only-locked table	Data pages: 80	Data pages: 50 Leaf pages: 80	*ncl_ix* leaf pages: 90 *pubid_ix* leaf pages: 75

Use of the *sorted_data* and *fillfactor* Options

The **sorted_data** option for **create index** is used when the data to be sorted is already in order by the index key. This allows **create clustered index** to skip the copy step while creating a clustered index. For example, if data that is bulk copied into a table is already in order by the clustered index key, creating an index with the **sorted_data** option creates the index without performing a sort. If the data does not need to be copied to new pages, the **fillfactor** is not applied.

The use of other **create index** options might still require copying, however. For more information, see “Creating an Index on Sorted Data” on page 23-4 of the *Performance and Tuning Guide*.

5 Locking During Query Processing

This chapter provides an overview of locking during query processing. For more information, see Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

This chapter contains the following sections:

Lock Types and Duration During Query Processing

The types and the duration of locks acquired during query processing depends on the type of command and the isolation level at which the command is run.

Lock Duration

The lock duration depends on the transaction isolation level and the type of query. Lock duration can be one of the following:

- Scan duration – Locks are released when the scan moves off the row or page, for row or page locks, or when the scan of the table completes, for table locks.
- Statement duration – Locks are released when the statement execution completes.
- Transaction duration – Locks are released when the transaction completes.

Lock Types

Table 5-1 shows the types of locks acquired by queries at different isolation levels, for each locking scheme for queries that do not use cursors. Table 5-2 shows information for cursor-based queries.

Table 5-1: Lock type and duration without cursors

Statement	Isolation Level	Locking Scheme	Table Lock	Data Page Lock	Index Page Lock	Data Row Lock	Duration
select **readtext** any type of scan	0	allpages datapages datarows	- - -	- - -	- - -	- - -	No locks are acquired.
	1 2 with **noholdlock** 3 with **noholdlock**	allpages datapages datarows	IS IS IS	S * -	S - -	- - *	* Depends on setting of **read committed with lock**. See "Locking for select Queries at Isolation Level 1" on page 5-5.
	2	allpages datapages datarows	IS IS IS	S S -	S - -	- - S	Locks are released at the end of the transaction. See "Isolation Level 2 and Allpages-Locked Tables" on page 5-6.
select via index scan	3 1 with **holdlock** 2 with **holdlock**	allpages datapages datarows	IS IS IS	S S -	S - -	- - S	Locks are released at the end of the transaction.
select via table scan	3 1 with **holdlock** 2 with **holdlock**	allpages datapages datarows	IS S S	S - -	- - -	- - -	Locks are released at the end of the transaction.
insert	0, 1, 2, 3	allpages datapages datarows	IX IX IX	X X -	X - -	- - X	Locks are released at the end of the transaction.
writetext	0, 1, 2, 3	allpages datapages datarows	IX IX IX	X X -	- - -	- - X	Locks are held on first text page or row; locks released at the end of the transaction.
delete **update** any type of scan	0, 1, 2	allpages datapages datarows	IX IX IX	U, X U, X -	U, X - -	- - U, X	"U" locks are released after the statement completes. "IX" and "X" locks are released at the end of the transaction.
delete **update** via index scan	3	allpages datapages datarows	IX IX IX	U, X U, X -	U, X - -	- - U, X	"U" locks are released after the statement completes. "IX" and "X" locks are released at the end of the transaction.
delete **update** via table scan	3	allpages datapages datarows	IX X X	U, X - -	- - -	- - -	Locks are released at the end of the transaction.
Key: IS intent shared, IX intent exclusive, S shared, U update, X exclusive							

Table 5-2: Lock type and duration with cursors

Statement	Isolation Level	Locking Scheme	Table Lock	Data Page Lock	Index Page Lock	Data Row Lock	Duration
select (without **for** clause) **select... for read only**	0	allpages datapages datarows	- - -	- - -	- - -	- - -	No locks are acquired.
	1 2 with **noholdlock** 3 with **noholdlock**	allpages datapages datarows	IS IS IS	S * -	S - -	- - *	* Depends on setting of **read committed with lock**. See "Locking for select Queries at Isolation Level 1" on page 5-5.
	2, 3 1 with **holdlock** 2 with **holdlock**	allpages datapages datarows	IS IS IS	S S -	S - -	- - S	Locks become transactional after the cursor moves out of the page/row. Locks are released at the end of the transaction.
select...for update	1	allpages datapages datarows	IX IX IX	U, X U, X -	X - -	- - U, X	"U" locks are released after the cursor moves out of the page/row. "IX" and "X" locks are released at the end of the transaction.
select...for update with shared	1	allpages datapages datarows	IX IX IX	S, X S, X -	X - -	- - S, X	"S" locks are released after the cursor moves out of page/row. "IX" and "X" locks are released at the end of the transaction.
select...for update	2, 3, 1 **holdlock** 2, **holdlock**	allpages datapages datarows	IX IX IX	U, X U, X -	X - -	- - U, X	Locks become transactional after the cursor moves out of the page/row. Locks are released at the end of the transaction.
select...for update with shared	2, 3 1 with **holdlock** 2 with **holdlock**	allpages datapages datarows	IX IX IX	S, X S, X -	X - -	- - S, X	Locks become transactional after the cursor moves out of the page/row. Locks are released at the end of the transaction.

Key: IS intent shared, IX intent exclusive, S shared, U update, X exclusive

Locking for *select* Queries at Isolation Level 1

When a **select** query on an allpages-locked table performs a table scan at transaction isolation level 1, it first acquires a shared intent lock on the table and then acquires a shared lock on the first data page. It locks the next data page, and drops the lock on the first page, so that the locks "walk through" the result set. As soon as the query completes, the lock on the last data page is released, and then the table-level lock is released. Similarly, during index scans on an allpages-locked table, overlapping locks are held as the scan descends the index to the data page. Locks are also held on the outer table of a join while matching rows from inner table are scanned.

select queries on data-only-locked tables first acquire a shared intent table lock. Locking behavior on the data pages and data rows is configurable with the parameter **read committed with lock**, as follows:

- If **read committed with lock** is set to 0 (the default) then **select** queries read the required column values with instant-duration page or row locks. The required column values or pointers for the row are read into memory, and the lock is released. Locks are not held on the outer tables of joins while rows from the inner tables are accessed. This reduces deadlocking and improves concurrency.

  If a **select** query needs to read a row that is locked with an incompatible lock, the query still blocks on that row until the incompatible lock is released. Setting **read committed with lock** to 0 does not affect the isolation level; only committed rows are returned to the user.

- If **read committed with lock** is set to 1, **select** queries acquire shared page locks on datapages-locked tables and shared row locks on datarows-locked tables. As described above, the lock on the first page or row is held, then the lock is acquired on the second page or row and the lock on the first page or row is dropped.

Cursors must be declared as read-only in order to avoid holding locks during scans when **read committed with lock** is set to 0. Any implicitly or explicitly updatable cursor on a data-only-locked table holds locks on the current page or row until the cursor moves off the row or page. When **read committed with lock** is set to 1, read-only cursors hold a shared page or row lock on the row at the cursor position.

read committed with lock does not affect locking behavior on allpages-locked tables. For information on setting the configuration parameter, see "read committed with lock" on page 16-8.

Table Scans and Isolation Levels

For queries at isolation level 1, see “Locking for select Queries at Isolation Level 1” on page 5-5.

Table Scans and Table Locks at Isolation Level 3

When a query performs a table scan at transaction isolation level 3 on a data-only-locked table, a shared or exclusive table lock provides phantom protection and reduces the locking overhead of maintaining a large number of row or page locks. On an allpages-locked table, an isolation level 3 scan first acquires a shared or exclusive intent table lock and then acquires and holds page-level locks until the transaction completes or until the lock promotion threshold is reached and a table lock can be granted.

Isolation Level 2 and Allpages-Locked Tables

Pre-11.9.2 versions of SQL Server and Adaptive Server supported isolation level 2 (repeatable reads) by also enforcing isolation level 3 (serializable reads). This restriction still applies to allpages-locked tables. If transaction level 2 is set in a session, and an allpages-locked table is included in a query, transaction isolation level 3 will also be applied on the allpages-locked tables. Transaction level 2 will be used on all data-only-locked tables in the session.

Update Locks Not Required for Some Indexed Deletes and Updates

All **update** and **delete** commands on an allpages-locked table first acquire an update lock on the data page and then change to an exclusive lock if the row meets the qualifications in the query.

Updates and delete commands on data-only-locked tables do not first acquire update locks when the following requirements are met:

- The query includes search arguments for every key in the index chosen by the query, so that the index unambiguously qualifies the row, and
- The query does not contain an **or** clause.

Updates and deletes that meet these requirements immediately acquire an exclusive lock on the data page or data row. This reduces lock overhead.

Locking During *or* Processing

In some cases, queries using **or** clauses are processed as a union of more than one query. Although some rows may match more than one of the **or** conditions, each row must be returned only once. Different indexes can be used for each **or** clause. If any of the clauses does not have a useful index, the query is performed using a table scan.

The table's locking scheme and the transaction isolation level affect how **or** processing is performed and the types and duration of locks that are held during the query.

Processing *or* Queries for Allpages-Locked Tables

If the **or** query uses index scans, and different **or** clauses might match the same rows, query processing retrieves the row IDs and matching key values from the index and stores them in a worktable, holding shared locks on the index pages containing the rows. When all row IDs have been retrieved, the worktable is sorted to remove duplicate values. Then, the worktable is scanned, and the row IDs are used to retrieve the data rows, acquiring shared locks on the data pages. The index and data page locks are released at the end of the statement (for transaction isolation level 1) or at the end of the transaction (for transaction isolation levels 2 and 3).

If the **or** query has no possibility of returning duplicate rows, no worktable sort is needed. At transaction isolation level 1, locks on the data pages are released as soon as the scan moves off the page.

Processing *or* Queries for Data-Only-Locked Tables

On data-only-locked tables, the type and duration of locks acquired for **or** queries where multiple clauses might match the same rows, depend on the transaction isolation level.

Processing *or* Queries at Transaction Isolation Levels 1 and 2

No locks are acquired on the index pages or rows of data-only-locked tables while row IDs are being retrieved from indexes and copied to a worktable. After the worktable is sorted to remove duplicate values, the data rows are requalified when the row IDs are used to read data from the table. If any rows were deleted, they are not returned. If any rows were updated, they are requalified by applying

the full set of query clauses to them. The locks are released when the row qualification completes, for isolation level 1, or at the end of the transaction, for isolation level 2.

Processing *or* Queries at Transaction Isolation Level 3

Transaction isolation level 3 requires serializable reads. At this isolation level, **or** queries obtain locks on the data pages or data rows during the first phase of **or** processing, as the worktable is being populated. These locks are held until the transaction completes. Requalification of rows is not required.

Row Locking and Lock Promotion

In versions prior to 11.9.2, Adaptive Server limits the overhead of managing large numbers of locks by promoting page locks to table locks when a process acquires more than a configurable number of page-level locks. Adaptive Server also supports promotion from row-level locks to table-level locks. Lock promotion settings for row-level locks are configured independently of settings for page-level locks.

The configuration parameters that manage page-level lock promotion have been renamed to indicate their function. The new names are: **page lock promotion HWM**, **page lock promotion LWM**, and **page lock promotion PCT**. Row lock promotion is configured using the **row lock promotion HWM**, **row lock promotion LWM**, and **row lock promotion PCT** parameters. For more information, see Chapter 16, "New and Changed Configuration Parameters."

➤ ***Note***

Lock promotion is always two-tiered: from page locks to table locks or from row locks to table locks. Row locks are never promoted to page locks.

6 How Locking Schemes Affect Object Sizes

This chapter discusses changes to page overhead and row format that affect the sizes of tables and indexes. It supplements the information in Chapter 6, "Determining or Estimating the Sizes of Tables and Indexes," in the *Performance and Tuning Guide*.

This chapter contains the following sections:

Page and Row Overhead in Data-Only-Locked Tables

The size of page headers and row overhead affects the number of rows that can be stored on a page in a data-only-locked table. In data-only-locked tables, there are 2002 bytes available on data and index pages. The maximum row size for data rows depends on whether there are variable-length columns in the table.

The available space on data and index pages (2016 bytes) and the maximum row size (1960 plus 2 bytes of overhead) for allpages-locked tables remain the same in version 11.9.2.

Page Overhead

Page overhead for data-only-locked tables is 46 bytes. (For allpages-locked tables, the overhead is 32 bytes.) This affects the number of rows per page for tables and indexes and the maximum size of user data per row.

Row Overhead

The row overhead for data-only-locked tables is:

- Fixed-length columns only – 6 bytes, plus the 2-byte row offset.
- Variable-length columns or columns that allow null values – 8 bytes, plus the 2-byte row offset.

Minimum Row Size

Data-only locked tables must allow room for each row to store a 6-byte forwarded row ID. If a data-only-locked table has rows shorter than 10 bytes, each row is padded to 10 bytes when it is inserted. This affects only data pages, and not indexes, and does not affect allpages-locked tables.

Maximum Row Size

The maximum size of a data row in a data-only locked table with only fixed-length columns is 1958 bytes of user data (plus 2 bytes of row overhead), compared to 1960 bytes of user data (plus 2 bytes of row overhead) for an allpages-locked table. For tables with variable-length columns, subtract 2 bytes for each variable-length column in the table.

Changes That Affect Index Size

Converting a table to a data-only locking scheme affects the sizes of its indexes.

Page Overhead

Page overhead on index pages is 46 bytes, which leaves 2002 bytes for index rows.

Clustered Indexes on Data-Only-Locked Tables

Clustered indexes on data-only-locked tables have a leaf level above the data level; the leaf level contains a row for each row in the table. This increases the size of the index, compared to the size of a clustered index on a the same table using allpages-locking. For allpages-locked tables, the level above the data pages is a sparse index, with just one row per data page.

Row Offset Tables

Indexes on data-only-locked tables have a row offset table, adding 2 bytes of overhead for each row on the page.

Suffix Compression and Elimination of Duplicate Keys

These factors reduce the size of indexes on data-only-locked tables:

- Suffix compression
- Elimination of duplicate keys

Suffix Compression on Non-Leaf Pages

Suffix compression is performed on the key values for character data in the index level just above the leaf level, only for indexes on data-only-locked tables. Instead of storing the entire key of the first row on the leaf page, only enough of the key to determine the difference between the last key on the previous page and the first key on the current page is stored. Suffix compression is used for keys on columns of the datatypes *char*, *varchar*, *nchar*, and *nvarchar*. By extension, suffix compression affects row IDs; when suffix compression is used, row IDs do not need to be stored on the non-leaf level.

Figure 6-1 shows how keys are stored without suffix compression and how suffix compression reduces the length of keys in the non-leaf level.

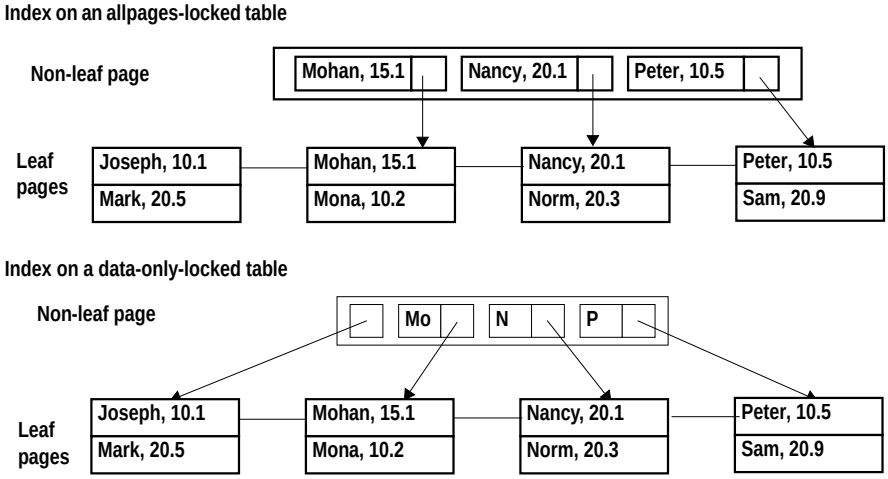

Figure 6-1: Suffix compression for data-only-locked tables

In Figure 6-1, the compressed suffix for the page with "Mohan" as the first key is determined by comparing it to the last key on the

previous page, “Mark”. Only two characters, “Mo”, are needed to distinguish the difference between “Mark” and “Mohan”. Similarly, for the pointer to the page storing “Peter”, the initial letter, “P”, is enough to distinguish it from “Norm”, the last key on the previous page.

Compressed key values may be much longer than the one- or two-letter compressed values shown in Figure 6-1. For example, the difference between “Parton” and “Partridge” would be “Partr”. For very long character strings, with only a few characters difference at the end, almost the entire string would need to be stored.

For composite indexes with character columns, compression can extend across the keys. For example, for an index on (*last_name*, *first_name*), the difference between “Elton, John” and “Elton, William” is “Elton, W”, and the difference between “Elton, John” and “Elton, Jon” is “Elton, Jon”.

Duplicate Values Stored Only Once Per Page

When an index on a data-only-locked table has duplicate keys, the key values are stored only once per index page, followed by a list of row IDs. The row IDs in a duplicate list are stored in increasing row ID order. For an index with many duplicates, this can greatly reduce the number of index pages.

Root and Leaf Pages for Very Small Tables

Indexes for small allpages-locked tables can contain a root page that points directly to the data pages so that there are no leaf pages in the

index. Indexes on data-only-locked tables, always have at least two pages: the root page and a leaf page, as shown in Figure 6-2.

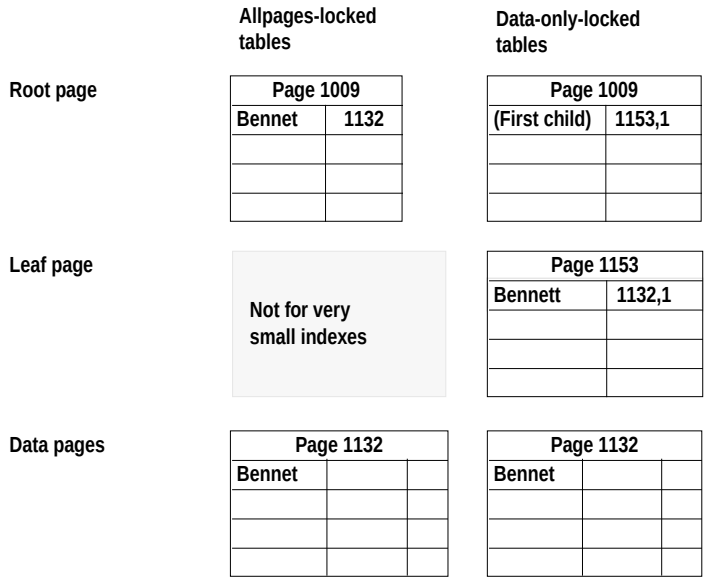

Figure 6-2: Data-only-locked tables always contain root and leaf levels

In allpages-locked tables, the root page, in rare circumstances, could change, requiring an update to the *root* column in *sysindexes*. For data-only-locked tables, the root page never changes.

Effects of Space Management Properties on Space Usage

The space management properties that affect space usage for data-only-locked tables and indexes are:

- **fillfactor**
- **exp_row_size**
- **reservepagegap**

➤ ***Note***

max_rows_per_page does not apply to data-only-locked tables.

fillfactor

Specifying the **fillfactor** during **create index** applies at the time the index is created. The **fillfactor** is not maintained during inserts to the table. If a **fillfactor** has been stored for an index using **sp_chgattribute**, this value is used when indexes are re-created due to **alter table...lock** commands and **reorg rebuild**. The main function of **fillfactor** is to allow space on the index pages, to reduce page splits. Very small **fillfactor** values can cause the storage space required for a table or an index to be significantly greater.

exp_row_size

Setting an expected row size for a table can increase the amount of storage required. If your tables have many rows that are shorter than the expected row size, setting this value and running **reorg rebuild** or changing the locking scheme will increase the storage space required for the table. However, the space usage for tables that formerly used **max_rows_per_page** should remain approximately the same. For more information, see "Reducing Row Forwarding with Expected Row Size" on page 4-1.

reservepagegap

Setting a **reservepagegap** for a table or an index leaves empty pages on extents that are allocated to the object when commands that perform extent allocation are executed. Setting **reservepagegap** to a low value increases the number of empty pages and spreads the data across more extents, so the additional space required is greatest immediately after a command such as **create index** or **reorg rebuild**. Row forwarding and inserts into the table fill in the reserved pages. For more information, see "Leaving Space for Forwarded Rows and Inserts" on page 4-8.

Changes to *sp_estspace*

The **sp_estspace** system procedure determines the locking scheme for a table and uses the appropriate calculations to estimate the space requirements for the table and its indexes.

Using Formulas to Calculate Space Requirements

Calculating the Sizes of Data-Only-Locked Tables

The formulas and examples that follow show how to calculate the sizes of tables and indexes. They are divided into two sets of steps:

- Steps 1–3 outline the calculations for a data-only-locked table. The example that follows Step 3 illustrates the computations on a table that has 9,000,000 rows.
- Steps 4–8 outline the calculations for computing the space required by an index, followed by an example using the 9,000,000-row table.

Step 1: Calculate the Data Row Size

Rows that store variable-length data require more overhead than rows that contain only fixed-length data, so there are two separate formulas for computing the size of a data row.

Use the first formula if all the columns are fixed length, and defined as NOT NULL. Use the second formula if the row contains variable-length columns or columns defined as NULL.

Fixed-Length Columns Only

Use this formula if the table contains only fixed-length columns:

```
      6   (Overhead)
+         Sum of bytes in all fixed-length columns
----------
          Data row size
```

Some Variable-Length Columns

Use this formula if the table contains variable-length columns or columns that allow null values:

```
      8   (Overhead)
+         Sum of bytes in all fixed-length columns
+         Sum of bytes in all variable-length columns
+         Number of variable-length columns * 2
----------
          Data row size
```

Step 2: Compute the Number of Data Pages

2002 / Data row size = Number of data rows per page

Number of rows / Rows per page = Number of data pages required

Step 3: Calculate Allocation Overhead and Total Pages

Allocation Overhead

Each table and each index on a table has an object allocation map (OAM). The OAM is stored on pages allocated to the table or index. A single OAM page holds allocation mapping for between 2,000 and 63,750 data pages or index pages. In most cases, the number of OAM pages required is close to the minimum value. To calculate the number of OAM pages for the table, use:

Number of reserved data pages / 63,750	=	Minimum OAM pages
Number of reserved data pages / 2000	=	Maximum OAM pages

Total Pages Needed

Finally, add the number of OAM pages to the earlier totals to determine the total number of pages required:

	Minimum	Maximum
Data pages	+	+
OAM pages	+	+
Total		

Example: Calculating the Size of a 9,000,000-Row Table

The following example computes the size of the data and clustered index for a table containing:

- 9,000,000 rows
- Sum of fixed-length columns = 100 bytes
- Sum of 2 variable-length columns = 50 bytes

Calculating the Data Row Size (Step 1)

The table contains variable-length columns.

	8	(Overhead)
+	100	Sum of bytes in all fixed-length columns
+	50	Sum of bytes in all variable-length columns
+	4	Number of variable-length columns * 2
	162	Data row size

Calculating the Number of Data Pages (Step 2)

In the first part of this step, the number of rows per page is rounded down:

2002 / 162	=	12 Data rows per page
9,000,000 / 12	=	750,000 Data pages

Calculating the Number of OAM Pages and Total Pages (Step 3)

750,000 / 63,750	=	12 (minimum)
750,000 / 2000	=	375 (maximum)

Total Pages Needed:

	Minimum	Maximum
Data pages	750000	750000
OAM pages	12	375
Total	750012	750375

Calculating the Sizes of Indexes for Data-Only-Locked Tables

Use these formulas for clustered and nonclustered indexes on data-only-locked tables.

Step 4: Calculate the Size of the Index Row

Index rows containing variable-length columns require more overhead than index rows containing only fixed-length values. Use

the first formula if all the keys are fixed length. Use the second formula if the keys include variable-length columns or allow null values.

Fixed-Length Keys Only

Use this formula if the index contains only fixed-length keys:

```
      9   (Overhead)
+         Sum of fixed-length keys
----------
          Size of index row
```

Some Variable-Length Keys

Use this formula if the index contains any variable-length keys:

```
      9   (Overhead)
+         Sum of length of fixed-length keys
+         Sum of length of variable-length keys
+         Number of variable-length keys * 2
----------
          Size of index row
```

Step 5: Calculate the Number of Leaf Pages in the Index

2002 / Size of index row = No. of rows per 0 page

No. of rows in table / No. of rows per page = No. of leaf pages

Step 6: Calculate the Number of Non-Leaf Pages in the Index

No. of leaf pages / No. of index rows per page = No. of pages at next level

If the number of index pages at the next level above is greater than 1, repeat the following division step, using the quotient as the next dividend, until the quotient equals 1, which means that you have reached the root level of the index:

No. of pages at last level / No. of index rows per page = No. of pages at next level

Step 7: Calculate the Total Number of Non-Leaf Index Pages

Add the number of pages at each level to determine the total number of pages in the index:

Index Levels	Pages
3	
2	+
1	+
0	+
	Total number of 2K pages used

Step 8: Calculate Allocation Overhead and Total Pages

Number of index pages / 63,750 = Minimum OAM pages

Number of index pages / 2000 = Maximum OAM pages

Total Pages Needed

Add the number of OAM pages to the total in Step 11 to determine the total number of index pages:

	Minimum	Maximum
Nonclustered index pages		
OAM pages	+	+
Total		

Example: Calculating the Size of a Nonclustered Index

The following example computes the size of a nonclustered index on the 9,000,000-row table used to show how to calculate the size of a table. There are two keys, one fixed length and one variable length.

- 9,000,000 rows
- Sum of fixed-length columns = 100 bytes
- Sum of 2 variable-length columns = 50 bytes

- Composite nonclustered index key:
  - Fixed length column, 4 bytes
  - Variable length column, 20 bytes

Calculate the Size of the Leaf Index Row (Step 4)

The index contains variable-length keys:

	9	(Overhead)
+	4	Sum of length of fixed-length keys
+	20	Sum of length of variable-length keys
+	2	Number of variable-length keys * 2
	35	Size of leaf index row

Calculate the Number of Leaf Pages (Step 5)

2002 / 35 = 57 Nonclustered index rows per page

9,000,000 / 57 = 157,895 leaf pages

Calculate the Number of Non-Leaf Pages (Step 6)

157895 / 57 = 2771	Index pages, level 1
2770 / 57 = 49	Index pages, level 2
48 / 57 =1	Index pages, level 3

Calculating Totals (Step 7)

Index Levels	Pages	Rows
3	1	49
2	49	2771
1	2771	157895
0	157895	9000000
	160716	

Calculating OAM Pages Needed (Step 8)

160713 / 63,750 = 3 (minimum)

160713 / 2000 = 81 (maximum)

Total Pages Needed

	Minimum	Maximum
Index pages	160716	160716
OAM pages	3	81
Total pages	160719	160797

7 New Locking Commands

This chapter documents new Transact-SQL syntax for commands that directly affect locking. Topics covered are:

Explicitly Locking a Table with the *lock table* Command

In pre-11.9.2 versions, a lock-promotion value on a table determines when a scan attempts to acquire a table lock. Initially, the scan acquires page locks. If the number of page locks crosses the lock-promotion threshold, the scan attempts to get a table lock. You cannot request a table lock programmatically.

Version 11.9.2 introduces a new **lock table** command that makes it possible to explicitly request a table lock for the duration of a transaction. This is useful in cases where:

- The same table will be scanned more than once in the same transaction, and each scan may need to acquire many page or row locks.
- It is known that a scan will exceed a table's lock-promotion threshold and will therefore attempt to escalate to a table lock in any event.

Locking a table from the outset is considerably more efficient than incurring the overhead of first acquiring a large number of individual row or page locks and then moving to a table lock, particularly when there is significant contention for the table.

The syntax for the **lock table** command is:

```
lock table table_name in {share | exclusive } mode
      [ wait [no_of_seconds] | nowait ]
```

The **wait** / **nowait** option allows you to specify how long the command waits to acquire a table lock if it is blocked (see "Using the wait / nowait Options in the lock table Command" on page 7-2).

The following considerations apply to the use of the **lock table** command:

- You must issue the **lock table** command within a transaction.
- You cannot use **lock table** on system tables.
- You can first use **lock table** to lock a table in **share** mode and then use it to upgrade the lock to **exclusive** mode.
- You can use separate **lock table** commands to lock multiple tables within the same transaction.
- There is no difference between a table locked with the **lock table** command and a table locked through lock promotion without the **lock table** command. Both are tracked in the *syslocks* table. The system procedure **sp_lock** reports the same information for both.

Using the *wait/nowait* Options in the *lock table* Command

Use the **wait** option in the **lock table** command to specify the length of time the command waits to acquire a table lock. For example, the following command, inside a transaction, sets a wait period of 2 seconds for acquiring a table lock on the *titles* table:

```
lock table titles in share mode wait 2
```

If the wait time expires before a table lock is acquired, the transaction proceeds, and row or page locking is used, exactly as it would have been without the lock table command, and the following informational message (error number 12207) is generated:

```
Could not acquire a lock within the specified wait
period. COMMAND level wait...
```

For an example of error handling during transactions, see “lock table” on page 12-37.

➤ ***Note***

If you use **lock table...wait** without specifying *no_of_seconds*, the command waits indefinitely for a lock.

You can set time limits on waiting for a lock at the session level and the system level. The wait period set with the **lock table** command overrides both of these, as described in the following section.

The **nowait** option is equivalent to the **wait** option with a 0-second wait: **lock table** either obtains a table lock immediately or generates the

informational message given above. If the lock is not acquired, the transaction proceeds as it would have without the **lock table** command.

Session-Level and System-Level Time Limits on Waiting for a Lock

You can use the new **set lock** command at either the session level or within a stored procedure to control the length of time a task will wait to acquire locks.

A System Administrator can use the new **sp_configure** option, **lock wait period**, to set a server-wide time limit on acquiring locks.

Setting a Session-Level Lock-Wait Limit

Use the **set lock wait** command to control the length of time that a command in a session or in a stored procedure will wait to acquire locks. The syntax is:

```
set lock { wait no_of_seconds | nowait }
```

The *no_of_seconds* parameter is entered as an integer. Thus, the following example sets a session-level time limit of 5 seconds on waiting for locks:

```
set lock wait 5
```

With one exception, if the **set lock wait** period expires before a command acquires a lock, the command fails, the transaction containing it is rolled back, and the following error message is generated:

```
Msg 12205, Level 17, State 2:
Server 'sagan', Line 1:
Could not acquire a lock within the specified wait
period. SESSION level wait period=300 seconds,
spid=12, lock type=shared page, dbid=9,
objid=2080010441, pageno=92300, rowno=0. Aborting
the transaction.
```

The exception to this occurs when the **lock table** command in a transaction sets a longer wait period than **set lock wait.** In this case, the transaction uses the **lock table** wait period before timing out, as described under "Using the wait/nowait Options in the lock table Command" on page 7-2.

The **set lock nowait** option is equivalent to the **set lock wait** option with a 0-second wait. If a command other than the **lock table** command is not

able to obtain a requested lock immediately, the command fails, its transaction is rolled back, and the preceding error message is generated.

If both a server-wide lock-wait limit and a session-level lock-wait limit are set, the session-level limit takes precedence. If no session-level wait period is set with **set lock wait**, the server-level wait period is used.

➤ ***Note***

Stored procedures do not use a wait period set at either the server level or the session level. The wait period for commands in a stored procedure is unbounded, unless you explicitly specify a time limit by using the **set lock wait** command inside the stored procedure.

Setting a Server-Wide lock-wait Limit

A System Administrator can configure a server-wide lock-wait limit with the configuration parameter **lock wait period**. The syntax is:

```
sp_configure "lock wait period" [, no_of_seconds]
```

If the lock-wait period expires before a command acquires a lock, unless there is an overriding **set lock wait** or **lock table** wait period, the command fails, the transaction containing it is rolled back, and the following error message is generated:

```
Msg 12205, Level 17, State 2:
Server 'wiz', Line 1:
Could not acquire a lock within the specified wait
period. SERVER level wait period=300 seconds,
spid=12, lock type=shared page, dbid=9,
objid=2080010441, pageno=92300, rowno=0. Aborting
the transaction.
```

A time limit entered through **set lock wait** or **lock table wait** overrides a server-level lock-wait period. Thus, for example, if the server-level wait period is 5 seconds and the session-level wait period is 10 seconds, an **update** command waits 10 seconds to acquire a lock before failing and aborting its transaction.

The default server-level lock-wait period is effectively "wait forever." To restore the default after setting a time-limited wait, you can use **sp_configure** to set the value of **lock wait period** as follows:

```
sp_configure "lock wait period", 0, "default"
```

Information on the Number of lock-wait Timeouts

The **sp_sysmon** system procedure reports on the number of times tasks waiting for locks did not acquire the lock within the specified period. See "Lock Timeout Information" on page 20-3.

Readpast Locking for Queue Processing

Readpast locking is an option available for the **select** and **readtext** commands and the data modification commands **insert**, **update**, **delete**, and **writetext**. It instructs a command to silently skip all incompatible locks it encounters, without blocking, terminating, or generating a message. It is primarily used when the rows of a table constitute a queue. In such a case, a number of tasks may access the table to process the queued rows, which could, for example, represent queued customers or customer orders. A given task is not concerned with processing a specific member of the queue, but with processing any available members of the queue that meet its selection criteria.

Readpast Syntax

The syntax for using readpast locking is:

```
{ select | insert | update | delete } ...
       from tablename [ holdlock | noholdlock ]
           [ readpast ] [ shared ]
           ...

readtext [[database.]owner.]table_name.column_name
   text_pointer offset size
   [holdlock | noholdlock]  [shared] [readpast]
       ...

writetext [[database.]owner.]table_name.column_name
   text_pointer [readpast] [with log] data
```

Incompatible Locks During Readpast Queries

For **select** and **readtext** commands, incompatible locks are exclusive locks. Therefore, **select** and **readpast** commands can access any rows or pages on which shared or update locks are held.

For **delete**, **update** and **writetext** commands, any type of page or row lock is incompatible, so that:

- All rows with shared, update, or exclusive row locks are skipped in datarows-locked tables, and
- All pages with shared, update, or exclusive locks are skipped in datapages-locked tables.

All commands specifying **readpast** block if there is an exclusive table lock, except **select** commands executed at transaction isolation level 0.

Allpages-Locked Tables and Readpast Queries

If the **readpast** option is specified for an allpages-locked table, the **readpast** option is ignored. The command operates at the isolation level specified for the command or session:

- If the isolation level is 0, dirty reads are performed, and the command returns values from locked rows, and does not block.
- If the isolation level is 1 or 3, the command blocks when pages with incompatible locks must be read.

Effects of Isolation Levels *select* Queries with Readpast

Readpast locking is designed to be used at transaction isolation level 1 or 2.

Session-Level Transaction Isolation Levels and Readpast

For data-only-locked tables, the effects of **readpast** on a table in a **select** command are shown in Table 7-1.

Table 7-1: Session-level isolation level and the use of readpast

Session Isolation Level	Effects
0, **read uncommitted** (dirty reads)	**readpast** is ignored, and rows containing uncommitted transactions are returned to the user. A warning message is printed.
1, **read committed**	Rows or pages with incompatible locks are skipped;, no locks are held on the rows or pages read
2, **repeatable read**	Rows or pages with incompatible locks skipped; shared locks are held on all rows or pages that are read until the end of the statement or transaction.

Table 7-1: Session-level isolation level and the use of readpast (continued)

Session Isolation Level	Effects
3, **serializable**	**readpast** is ignored, and the command executes at level 3. The command blocks on any rows or pages with incompatible locks.

Query-Level Isolation Levels and Readpast

If **select** commands that specify **readpast** also include any of the following clauses, the commands fail and display error messages:

- The **at isolation** clause, specifying **0** or **read uncommitted**
- The **at isolation** clause, specifying **3** or **serializable**
- The **holdlock** keyword on the same table

If a **select** query that specifies **readpast** also specifies **at isolation 2** or **at isolation repeatable read**, shared locks are held on the **readpast** table or tables until the statement or transaction completes.

readtext commands that include **readpast** and that specify **at isolation read uncommitted** automatically run at isolation level 0 after issuing a warning message.

Data Modification Commands with *readpast* and Isolation Levels

If the transaction isolation level for a session is 0, the **delete**, **update**, and **writetext** commands that use **readpast** do not issue warning messages.

- For datapages-locked tables, these commands modify all rows on all pages that are not locked with incompatible locks.
- For datarows-locked tables, they affect all rows that are not locked with incompatible locks.

If the transaction isolation level for a session is 3 (serializable reads), the **delete**, **update**, and **writetext** commands that use **readpast** automatically block when they encounter a row or page with an incompatible lock.

At transaction isolation level 2 (serializable reads), the **delete**, **update**, and **writetext** commands:

- Modify all rows on all pages that are not locked with incompatible locks.
- For datarows-locked tables, they affect all rows that are not locked with incompatible locks.

text and *image* Columns and Readpast

If a **select** command with the **readpast** option encounters a text column that has an incompatible lock on it, readpast locking retrieves the row, but returns the text column with a value of **null**. No distinction is made, in this case, between a text column containing a null value and a null value returned because the column is locked.

If an **update** command with the **readpast** option applies to two or more text columns, and the first text column checked has an incompatible lock on it, readpast locking skips the row. If the column does not have an incompatible lock, the command acquires a lock and modifies the column. Then, if any subsequent text column in the row has an incompatible lock on it, the command blocks until it can obtain a lock and modify the column.

A **delete** command with the **readpast** option skips the row if any of the text columns in the row have an incompatible lock.

Readpast-Locking Examples

The following examples illustrate readpast locking.

To skip all rows that have exclusive locks on them:

```
select * from titles readpast
```

To update only rows that are not locked by another session:

```
update titles
    set price = price * 1.1
    from titles readpast
```

To use readpast locking on the *titles* table but not on the *authors* or *titleauthor* table:

```
select *
    from titles readpast, authors, titleauthor
    where titles.title_id = titleauthor.title_id
    and authors.au_id = titleauthor.au_id
```

To delete only rows that are not locked in the *stores* table, but to allow the scan to block on the *authors* table:

```
delete stores from stores readpast, authors
where stores.city = authors.city
```

Repeatable Reads Isolation Level

In previous versions of SQL Server and Adaptive Server, transaction isolation level 2 (repeatable reads) was supported by requiring users to specify transaction isolation level 3 (serializable reads). In version 11.9.2, transaction isolation level 2 is explicitly provided for data-only-locked tables.

➤ ***Note***

If you use transaction isolation level 2 (repeatable reads) on allpages-locked tables, isolation level 3 (serializable reads) is also enforced.

A transaction performing repeatable reads locks all rows or pages read during the transaction. After one query in the transaction has read rows, no other transaction can update or delete the rows until the repeatable reads transaction completes. However, repeatable-reads transactions do not provide phantom protection by performing range locking, as serializable transactions do. Other transactions can insert values that can be read by the repeatable-reads transaction and can update rows so that they match the search criteria of the repeatable-reads transaction.

To enforce repeatable reads at a session level, use:

```
set transaction isolation level 2
```

or

```
set transaction isolation level repeatable read
```

To enforce transaction isolation level 2 from a query, use:

```
select title_id, price, advance
from titles
at isolation 2
```

or

```
select title_id, price, advance
from titles
at isolation repeatable read
```

Transaction isolation level 2 is not available at the object level.

For more information on locks held during isolation level 2 reads, see Chapter 5, “Locking During Query Processing,” in this manual. For more information on transaction isolation levels, see Chapter 5, “Locking in Adaptive Server,” in the *Performance and Tuning Guide*.

Statistics and Query Optimization Changes

8 Statistics Enhancements

This chapter explains how statistics are stored and displayed in version 11.9.2. Topics include:

For more information on managing statistics, see Chapter 9, "Managing Table and Index Statistics." For examples of how these statistics are used in query processing, see Chapter 10, "Query Processing Changes." Additional examples are presented in Chapter 11, "Changes to showplan and dbcc traceon(302) Output."

Overview of Statistics Enhancements

- Two new system tables, *systabstats* and *sysstatistics* store statistics that are used to optimize queries. See "New System Tables Store Statistics" on page 8-2 for more information.
- A new utility, **optdiag**, displays statistics and allows some of them to be edited. Users should never directly update the *systabstats* and *sysstatistics* tables using Transact-SQL. See "Viewing Statistics with the optdiag Utility" on page 8-6 and "Changing Statistics with optdiag" on page 8-24.
- Changes to the syntax for **update statistics** allow you to create statistics for columns other than the leading columns of an index, and the optimizer now uses these statistics to improve query costing. See Chapter 9, "Managing Table and Index Statistics," for more information.

➤ ***Note***

If you load a database from a pre-11.9 version of SQL Server or Adaptive Server, an internal upgrade process copies the distribution page information for each index to the new system tables and then removes the distribution page. Since this process does not scan tables to determine values for the new statistics provided in version 11.9.2, you should run **update statistics** to generate these new statistics for your tables. Otherwise, default values are used for new statistics.

New System Tables Store Statistics

The *systabstats* and *sysstatistics* tables store statistics for all tables, indexes, and unindexed columns. Some of the statistics stored in these tables were formerly stored on the distribution page that was part of each index. Other statistics were stored in other structures in the database, such as OAM pages. New types of statistics are gathered and stored in these tables and are used to improve the query optimizer's cost estimates.

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

The *systabstats* table 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 "systabstats" on page A-8 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 8-13.

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 8-18 for more information.

See "sysstatistics" on page A-7 for more information.

How Statistics Data Is Created and Maintained

The information stored in *systabstats* and *sysstatistics* is affected by data definition language (DDL). Some data modification language

also affects *systabstats*. Table 8-1 summarizes how DDL affects the *systabstats* and *sysstatistics* tables.

Table 8-1: Effects of DDL on systabstats and sysstatistics

Command	Effect on *systabstats*	Effect on *sysstatistics*
alter table...lock	Changes values to reflect the underlying 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.
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.
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.

Table 8-1: Effects of DDL on systabstats and sysstatistics (continued)

Command	Effect on *systabstats*	Effect on *sysstatistics*
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 causes index re-creation.
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		
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 task.

If you need to query *systabstats* directly, you can flush the in-memory statistics to *systabstats* with the system procedure **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.

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.

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] [-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 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 Chapter 15, "optdiag Utility," for more information on the **optdiag** command. The follow 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 8-2 describes the output.

Table 8-2: 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 8-3 describes the rows in the report.

Table 8-3: 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.

Table 8-3: Table statistics (continued)

Row Label	Information Provided
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 8-9.
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.
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 is used to help determine the effectiveness of large I/O for table scans and range scans. This value is updated only when you run **update statistics.**

For data-only-locked tables, the Data Page CR (cluster ratio) count reflects the number of extent jumps incurred during an OAM scan. It does not include the I/O needed to read the OAM and allocation pages.

For allpages-locked tables, the Data Page CR count is the total number of extent I/Os required to read the entire table following the page chain. If the table is fragmented because page splits cause the page chain to cross extents, the Data Page CR count is incremented. The value is used compute the data page cluster ratio, which is used

during query processing to estimate the additional extent I/Os required to scan the table.

In Figure 8-1, the page chain crosses extents, increasing the data page CR count for the table.

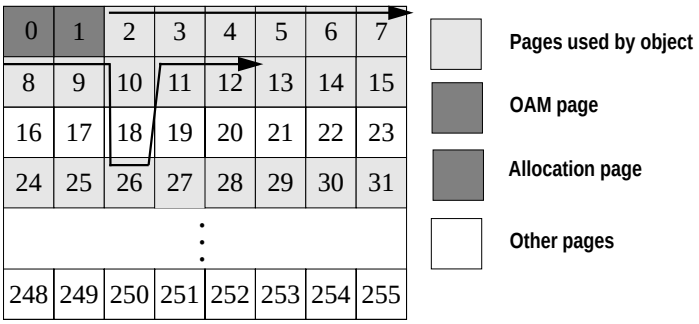

Figure 8-1: Page chain crossing extents in an allpages-locked table

Table Cluster Ratio

The "Derived statistics" in the table-level section reports the data page cluster ratio for the table, derived from the "Data Page CR count", and the number of extents in use by the table. Table 8-4 describes the output.

Table 8-4: Cluster ratio for a table

Row Label	Information Provided
Data page cluster ratio	This ratio 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. The data page cluster ratio is used to estimate the effectiveness of large I/O.

For allpages-locked tables, the data page cluster ratio measures how well pages are clustered on extents, when the table is read in page-chain order. A cluster ratio of 1.0 indicates perfect clustering.

For data-only-locked tables, the data page cluster ratio measure how well the pages are packed on the extents. If extents contain unused pages, the data page cluster ratio is lower. A cluster ratio of 1.0 indicates complete packing of extents.

For an example of how the data page cluster ratio is used, see "Costing for Clustered Indexes on Data-Only-Locked Tables" on page 10-3.

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 8-5 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 8-5: 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 ratio for accessing a table using the index. See "Index Cluster Ratios" on page 8-12.
Index page CR count	A counter used to compute the index page cluster ratio. See "Index Cluster Ratios" on page 8-12.

Table 8-5: Index statistics (continued)

Row Label	Information Provided
Data row CR count	A counter used to compute the data row cluster ratio See "Index Cluster Ratios" on page 8-12.
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 Cluster Ratios

The "Derived statistics" in the index-level section are based on the "CR count" values shown in "Index Statistics" on page 8-11. Table 8-6 describes the output.

Table 8-6: 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 clustering 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 clustering 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 clustering of rows on data pages.

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.

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 and 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 indexes 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** would improve performance, especially if many queries perform covered scans.

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.

For an example of how the data row cluster ratio and the index page cluster ratio are used to optimize queries, see "Costing for Clustered Indexes on Data-Only-Locked Tables" on page 10-3.

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 8-7 for each prefix subset of the index keys. For columns A, B, C, D, the prefix subsets are:
  - A
  - A, B
  - A, B, C
  - A, B, C, D

  The set of columns A, D or B, C are not prefix subsets.

  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 8-7 describes the output.

Table 8-7: 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 8-16.
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 8-16.
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 value 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 8-19 for information on range and frequency cells.
- The "Total density" measures the duplicate values for all columns, including those represented by 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 the 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 8-17 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. The **update index statistics** command generates statistics for all columns in indexes.

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 8-8 shows the default values that are used.

Table 8-8: Density approximations for unknown search arguments

Operation Type	Operator	Density Approximation
Equality	**=**	Total density, if statistics are available for the column, or 10%
Open-ended range	**<**, **<=**, **>**, or **>=**	33%
Closed range	**between**	25%

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 8-26 for information on using **optdiag** to supply selectivity values.

Histogram Displays

Histograms store information about the distribution of values in a column. Table 8-9 shows the commands that create and update histograms and which columns are affected.

Table 8-9: 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 8-10.

Table 8-10: 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
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 8-11.

Table 8-11: 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 this step 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, the fourth cell includes Charlotte (the upper bound), but excludes Boston (the lower bound). The weight for this step is .0526, indicating that 5.26 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:

- 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, or
- 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 histograms 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      "
```

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 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 1 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.

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 for the statistics. 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    <=    0x41504f204d69616d68ffffffffffffffffffffff
# 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:              0.200000000
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

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 20's, 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 8-32 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

The syntax for **optdiag** simulate mode 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] [-Tflag_value]
       [-z language] [-J client_charset]
       [-a display_charset]
```

For example, 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 3 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 **max parallel degree** and **max scan parallel degree** configuration parameters:

```
Configuration Parameters:

     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

Simulated statistics can be loaded into the original database, or into a database created solely for performing "what-if" analysis on queries.

- 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.

- 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.

In order 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 8-34.

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 query in a procedure

Dropping Simulated Statistics

Loading simulated statistics adds rows describing cache configuration to the *sysstatistics* table in the *master* database. To remove these statistics, use the command **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 recreate just 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

The command **set statistics simulate on** tells the optimizer to optimize queries using simulated statistics. 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.

9 Managing Table and Index Statistics

Adaptive Server version 11.9.2 provides more statistics for query optimization and more control over those statistics than previous versions.

This chapter contains the following sections:

Summary of Changes That Affect Statistics Management

The following changes to Adaptive Server 11.9.2 affect statistics management:

- In earlier versions of SQL Server and Adaptive Server, the number of steps for histograms was limited by the requirement that all statistics fit on a single page that was part of the index. In version 11.9.2, statistics are kept in system tables. You can specify the number of steps for a histogram with an **update statistics** command or a **create index** command.
- Additional syntax for **update statistics** allows you to create statistics for columns that are not the leading columns of an index.

➤ ***Note***

The **update all statistics** command now creates statistics for all columns in a table rather than just for the leading columns. It also updates partition statistics, as in version 11.5. If your statistics maintenance includes **update all statistics**, determine whether you want to replace **update all statistics** with individual **update statistics** and **update partition statistics** commands.

- The **drop index** command no longer deletes all information about the index when the index is dropped. The column statistics information remains in the *sysstatistics* table and can still be used by the optimizer for estimating the cost of any **where** or **having** clause on the column.

- The **update statistics** and **create index** commands check the number of steps in a distribution histogram for a column and use the existing number of steps as the default value.
- To re-create an index without overwriting the statistics for the index, you can specify the number of steps as 0.
- When Adaptive Server reads the statistics for a table during query optimization, the statistics are stored in the procedure cache. The statistics are stored in the cache only as long as they are needed by the query.

➤ ***Note***

If you load a dump of a pre-11.9 database, the information on the distribution page is converted and stored in *sysstatistics*, using the same number of steps that were used on the distribution page. The distribution page is deallocated. Running **update statistics** produces more complete and accurate statistics than the upgrade process on a loaded database, so you should run **update statistics** as soon as it is convenient.

Number of Steps Is Maintained After Upgrade

During upgrade, distribution page information from existing indexes is used to generate *sysstatistics* data for each indexed column. The number of steps in the pre-upgrade distribution page is used to determine the number of steps to be used for the histogram in the *sysstatistics* table. The **update statistics** and **create index** commands use the existing number of steps as a default value, so the number of steps for a table's histogram only changes:

- If you specify a different number of steps with **update statistics** or **create index.** This number of steps becomes the default.
- After a **delete statistics** command is issued. This command removes a histogram from *sysstatistics*. Subsequent **create index** commands use the default number of steps (20), unless the number of steps is specified.

The number of steps available on the distribution page for an index depends on the key size. For integer values, approximately 180 steps fit on a distribution page, but datatypes that require more bytes of storage result in few steps on the 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

The default value of 20 steps 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

See "Choosing the Number of Steps for Highly Duplicated Values" on page 8-24 for more information.

Column Statistics vs. Index Statistics

A major difference between version 11.9.2 and previous versions is that statistics in 11.9.2 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

The change from index-level statistics to column-level statistics in version 11.9.2 provides query tuners an opportunity to improve the performance of many queries. The optimizer can now 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 optimizers 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 11-14 for information on how additional column statistics can be used in query optimization.

Identifying 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. See "New and Changed dbcc traceon(302) Messages" on page 11-2 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 statistics 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
```

Scan Types, Sort Requirements, and Locking During *update statistics*

Table 9-1 shows the types of scans performed during **update statistics**, the types of locks acquired, and when sorts are needed.

Table 9-1: 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		
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 Nonleading 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 15, “Parallel Sorting,” in the *Performance and Tuning Guide*, 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 15, "Parallel Sorting," in the *Performance and Tuning Guide* 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 nonleading 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 Chapter 22, "Distributing Engine Resources Between Tasks," in the *Performance and Tuning Guide*.
- 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 command deletes the statistics for the *price* column in the *titles* table:

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

➤ ***Note***

The **delete statistic**s 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 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 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.

10 Query Processing Changes

This chapter contains the following sections:

Costing OAM Scans for Data-Only-Locked Tables

Tables that use a data-only locking scheme do not have page chains like allpages-locked tables. To perform a table scan on a data-only-locked table, Adaptive Server:

- Reads the OAM (Object Allocation Map) page(s) for the table
- Uses the pointers on the OAM page to access the allocation pages
- Uses the information on the allocation pages to locate the extents used by the table
- Performs either large I/O or 2K I/O on the pages in the extent

Figure 10-1 shows the pointers from OAM pages to allocation pages and from allocation pages to extents.

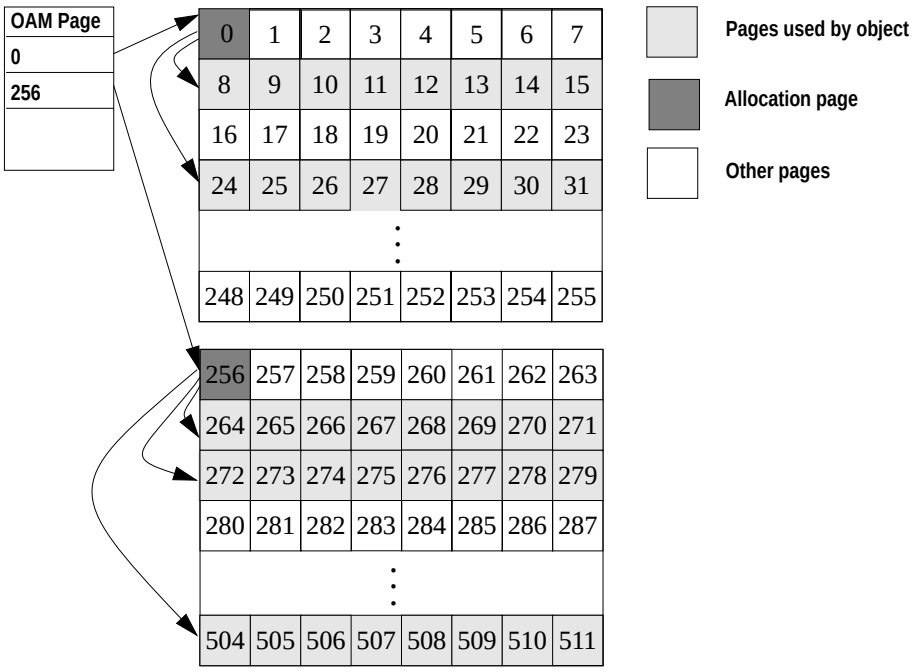

Figure 10-1: Sequence of pointers for OAM scans

The formula for computing the cost of an OAM scan with 2K I/O is:

```
Num_OAM_pages + Num_allocation_pages + Num_pages
```

The formula for computing the cost of an OAM scan with large I/O is:

```
Num_OAM_pages + Num_allocation_pages +
        Num_pages/Pages_per_IO
```

If a query on a data-only-locked table uses a parallel hash-based scan, the query hashes on either the extent number or the allocation page number, rather than hashing on the page number. The choice of whether to hash on the allocation page or the extent number is a cost-based decision made by the optimizer. Both methods can reduce the cost of performing parallel queries on unpartitioned tables. Queries that choose a serial scan on an allpages-locked table may use one of the new hash-based scan methods if the table is converted to data-only locking.

Allpages-locked tables do not use OAM scans. They are scanned by following the table's page chain, as described in Chapter 3, "Data Storage," in the *Performance and Tuning Guide*.

Costing for Clustered Indexes on Data-Only-Locked Tables

When a clustered index is created on a data-only-locked table, the index has the same structure as a nonclustered index. It has a leaf level above the data rows. The leaf level contains a key value and pointer for each row in the table.

As rows are inserted into allpages-locked tables, they are maintained in clustered index order by splitting pages and linking pages in key order. For data-only-locked tables, Adaptive Server attempts to keep rows in clustered key order, but never splits data pages. Rows are inserted on pages with adjacent keys when there is room. When there is no room, allocation algorithms attempt to place new rows on pages in physical proximity to the adjacent key in the index's key order. In addition, row forwarding may be required if rows grow too long to fit on the page on which they were originally inserted. If possible, forwarded rows are placed near the original location.

To cost queries using clustered indexes on data-only-locked tables, the optimizer uses statistics to estimate the number of rows to be returned by the query, and then estimates the number of logical I/Os required by using cluster ratios.

Covered index scans can be performed using clustered indexes on data-only-locked tables, since these indexes contain a leaf level above the data level.

Costing for Noncovered Scans

The number of I/Os for a noncovered index scan is estimated as:

- The index height, not including the leaf level, plus
- The number of index leaf pages qualified, estimated from histogram values, plus
- The number of data pages qualified, estimated by the number of rows qualified and the data row cluster ratio for the index, plus
- A proportional number of forwarded rows; that is, if a scan returns 10 percent of a table that has 80 forwarded rows, an additional 8 I/Os are estimated for the scan.

If large I/O is configured for the table, the data page cluster ratio is used to determine its effectiveness. If large I/O is available for the index, the index page cluster ratio is used to determine its effectiveness.

Costing for Covered Scans

Because there is now a full leaf level above the data pages, covered queries can be performed on data-only-locked tables with clustered indexes. The costing for these scans is the same as the costing for any nonclustered index scan.

The number of I/Os for a covered query is estimated as:

- The index height, not including the leaf level, plus
- The number of index leaf pages that qualify, estimated by the number of rows qualified.

If the index uses a cache configured for large I/O, the index page cluster ratio is used to determine whether large I/O is effective for this index.

The final physical I/O estimate for the scan is derived from a formula that uses the number of logical I/Os and the available space in the buffer pools used by the table or index.

Estimating the Cost of Forwarded Rows

Data-only-locked tables require that all row IDs remain fixed, except during some **reorg** commands or when rebuilding clustered indexes, since creating a clustered index copies the data rows to new pages. When the length of a row increases so that it no longer fits on the page where it is stored, the fixed row ID is maintained by:

- Inserting the row on a page where there is space
- Replacing the original row with a pointer to the new location

This process is called **row forwarding**, and the resulting row is called a **forwarded row**. Figure 10-2 shows the home location of a forwarded row pointing to the page where the row was inserted.

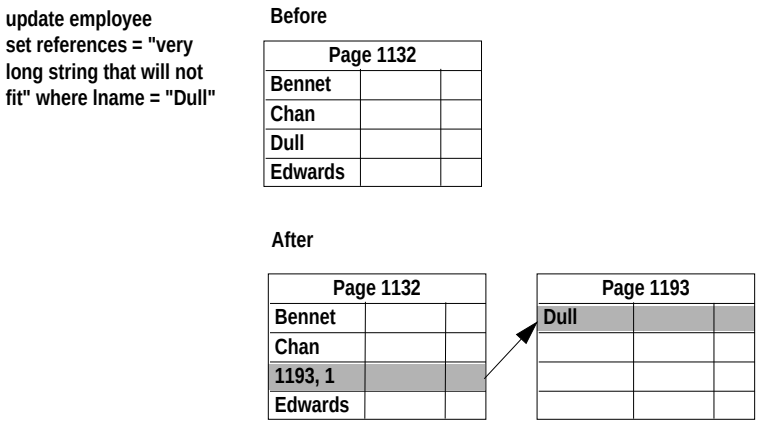

Figure 10-2: Original row stores pointer to forwarded row location

Query Processing Costs and Forwarded Rows

When a query needs to read a forwarded row, it performs one I/O to read the original location of the row, and a second I/O to read the forwarded location, increasing the cost of processing tables with forwarded rows. As part of the statistics maintained on data-only-locked tables, Adaptive Server maintains a count of forwarded rows. When the optimizer estimates the cost of I/O for data-only-locked tables, it adds additional I/O cost for forwarded rows. For example, if a table has 1000 pages and 80 forwarded rows, the optimizer estimates that 1080 logical I/Os are required for a table scan.

For a query that accesses only part of a table that contains forwarded rows, the cost of accessing forward rows is proportional to the percentage of the table that is accessed. For example, if a query needs to access 10 percent of a table with 80 forwarded rows, the estimated I/O cost for forwarded rows is 8 additional I/Os.

Allpages-locked tables do not use row forwarding.

Enhanced Large I/O Costing and Performance Features

Large I/O can greatly improve the performance of queries that need to access a large number of pages. Adaptive Server version 11.9.2

introduces these performance enhancements to improve large I/O costing:

- Using data and index page cluster ratios to improve the optimizer's cost estimates for queries that can use large I/O
- Using large I/O for both the index and data pages for noncovering index scans

Using Page Cluster Ratios to Improve Large I/O Estimates

If the table or index uses a cache with a buffer pool configured for large I/O, large I/O can be performed:

- On the data pages, for table scans and for range scans using clustered indexes
- On the leaf level of a nonclustered index, for covered queries

In pre-11.9.2 versions of SQL Server and Adaptive Server, the optimizer uses an estimate of 0.8 as the page cluster ratio for data and index pages. An estimated data page cluster ratio of 0.8 is pessimistic for a newly created clustered index, where cluster ratios are likely to be 1.0, and optimistic for indexes where many page splits have taken place, because cluster ratios can be significantly lower. Similarly, an index page cluster ratio of 0.8 is pessimistic for newly created nonclustered indexes, because the cluster ratio for leaf pages is near 1.0 immediately after the index is created.

Cluster ratio information is maintained for tables and indexes to improve large I/O estimates. **optdiag** displays statistics about the cluster ratio for tables and indexes. These values are:

- The data page cluster ratio, which measures how well-clustered the data pages are, with respect to extents
- The index page cluster ratio, which measures how well-clustered the index pages are, with respect to extents

➤ ***Note***

The data row cluster ratio, another cluster ratio used by query optimization, is used to cost the number of data pages that need to be accessed during scans using a particular index. It is not used in large I/O costing.

Data is kept on the cluster ratio for tables and indexes for all locking schemes, and the optimizer assesses the clustering for tables with all locking schemes.

Large I/O and Cache Strategy for Index and Data Pages

The **prefetch** clause in a **select** statement instructs Adaptive Server to use large I/O whenever possible for a table in a query. You can also specify a choice of cache strategy by specifying **lru** or **mru** clause. See Chapter 10, "Advanced Optimizing Techniques," in the *Performance and Tuning Guide* for more information.

In previous versions, the **prefetch** and **lru** or **mru** clauses applied to:

- Nonclustered index leaf pages, for a covered index scan
- Data pages, for a table scan, clustered index scan, or noncovered, nonclustered index scan

In version 11.9.2, specifying an I/O size or cache strategy in a query that performs a noncovered nonclustered index scan applies large I/O and the specified strategy to both the leaf-level index pages and the data pages. **showplan** now prints messages displaying the I/O size and cache strategy used for both index and data pages. The buffer replacement strategy is also displayed for both index and data pages. This query specifies 16K I/O and MRU cache strategy:

```
select type, avg(advance)
from titles (index type_price prefetch 16 mru)
where price > 100
group by type
```

This **showplan** output shows messages for the leaf and data pages:

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

Concurrency Optimization for Small Tables

For data-only-locked tables of 15 pages or fewer, Adaptive Server does not consider a table scan if there is a useful index on the table. Instead, it always chooses the cheapest index that matches any optimizable search argument in the query. The locking required for an index scan provides higher concurrency and reduces the chance of deadlocks, although slightly more I/O may be required than for a table scan. Small, frequently used tables and their indexes are likely to be kept in cache, incurring only additional logical I/O rather than increased physical I/O.

If concurrency on very small tables is not an issue, and you want to optimize the I/O instead, you can disable this optimization with

sp_chgattribute. This command turns off concurrency optimization for a table:

```
sp_chgattribute tiny_lookup_table,
    "concurrency_opt_threshold", 0
```

With concurrency optimization disabled, the optimizer can choose table scans when they require fewer I/Os.

You can also increase the concurrency optimization threshold for a table. This command sets the concurrency optimization threshold for a table to 30 pages:

```
sp_chgattribute small_lookup_table,
    "concurrency_opt_threshold", 30
```

The maximum value for the concurrency optimization threshold is 32,767. The current setting is stored in *systabstats.conopt_thld* and is printed as part of **optdiag** output.

Changing the Locking Scheme

Concurrency optimization affects only data-only-locked tables. Table 10-1 shows the effect of changing the locking scheme.

Table 10-1: Effects of alter table on concurrency optimization settings

Changing Locking Scheme From	Effect On Stored Value
Allpages to data-only	Set to 15, the default
Data-only to allpages	Set to 0
One data-only scheme to another	Configured value retained

Indexing Requirements for Cursors on Data-Only-Locked Tables

On allpages-locked tables, cursors require that a table have a unique index that can be used to position the scan for:

- Transaction isolation level 0 scans (dirty reads)
- Updatable cursors – not only must there be a unique index, but the scan must use an index whose columns are not included in the **for update** clause of the cursor.

These requirements mean that in some cases, IDENTITY columns must be added to indexes to make them unique, or that the optimizer might be forced to choose a suboptimal query plan for a cursor

query. These requirements are still true for allpages-locked tables in version 11.9.2, but do not apply to data-only-locked tables.

In data-only-locked tables, fixed row IDs are used to position cursor scans, so unique indexes are not required for dirty reads or updatable cursors.

Table Scans to Avoid the Halloween Problem

The Halloween problem is an update anomaly that can occur when a client using a cursor updates a column of the cursor result set row, and that column defines the order in which the rows are returned from the base table. For example, if a cursor were to use an index on *last_name*, *first_name*, and updates one of these columns, the row could appear in the result set a second time.

To avoid the Halloween problem on data-only-locked tables, Adaptive Server chooses a table scan when the columns from an otherwise useful index are included in the column list of a **for update** clause.

Allpages-locked tables still require a unique index that does not include columns in the column list of the **for update** clause.

For implicitly updatable cursors declared without a **for update** clause, and for cursors where the column list in the **for update** clause is empty, cursors that update a column in the index used by the cursor may encounter the Halloween problem.

Delay of Cursor Compilation Until Cursor Open

In pre-11.9.2 versions of SQL Server and Adaptive Server, a cursor is optimized and compiled at the time the cursor is declared. For cursors with local variables or parameters, this can result in a suboptimal query plan, since default values are used to estimate the number of rows that would be returned. The problem of poor plans is most common with client cursors, sent using Open Client™ Client-Library™ or Embedded SQL™.

In version 11.9.2, query optimization is delayed until the cursor is opened. The values of the variables are known at cursor open time, and query optimization may be considerably improved.

Visible Effects of Delayed Cursor Optimization

Besides improved query plans, the only user-visible changes are as follows:

- If you are using **showplan**, the plan is no longer displayed at the **declare cursor** statement, but at the **open** statement.
- The query is normalized at the **declare cursor** statement, so error messages about syntax errors, missing tables, and so forth, are reported at that time.
- Some error messages do not appear until the query is compiled and optimized, including those for cursors that require unique indexes. If error messages result at cursor open time (such as the lack of a needed index), the cursor must be deallocated and redeclared. You cannot just correct the problem and re-open the cursor.
- In earlier versions, changing isolation levels between cursor **declare** time and cursor **open** time generated an error message when the cursor was opened. In version 11.9.2, the following sequence of commands does not generate an error, and the cursor is compiled at transaction isolation level 0:

```
/*start at isolation level 1 */
declare curs1 cursor for
select t.title_id, au_lname
from titles t , authors a, titleauthor ta
where t.title_id = ta.title_id
and a.au_id = ta.au_id
go

set transaction isolation level 0
go
open curs1
go
fetch curs1
go
```

  Once the cursor has been opened at a particular isolation level, you must deallocate the cursor before changing isolation levels. The effects of changing isolation levels while the cursor is open are as follows:

  - Attempting to close and reopen the cursor at another isolation level fails with an error message.
  - Attempting to change isolation levels without closing and reopening the cursor has no effect on the isolation level in use and does not produce an error message.

➤ ***Note***

Delayed optimization does not apply to server cursors, that is, cursors that are declared in a stored procedure. Server cursors are compiled and optimized when they are declared.

Join Cursor Processing and Data Modifications

This section describes changes to cursor behavior in version 11.9.2 that may affect applications that use cursors. It also describes cursor positioning and modification rules for cursors on data-only-locked tables.

Updates and Deletes That Can Affect the Cursor Position

Two types of deletes and updates can affect the row at the cursor position:

- **Positioned** deletes and updates, using **delete...where current of** or **update...where current of** to change the row at the cursor position
- **Searched** deletes and updates, that is, any **delete** or **update** query that changes a value in the row at the cursor position, but without including a **where current of** clause.

Cursor Positioning After a *delete* or *update* Command Without Joins

The behavior of a **delete** or **update** command to the base table for a cursor on a single-table query depends on the type of modification, the locking scheme of the base table, and whether the clustered index of the base table is affected:

- A positioned **delete** or a searched **delete** that deletes the row at the cursor location positions the cursor to the next qualifying row on the table. This is true for both allpages-locked and data-only-locked tables. A subsequent positioned **update** or **delete** via this cursor is disallowed until the next **fetch** is done to position the cursor on the next row that qualifies.
- A searched or positioned **update** that does not change the position of the row leaves the current position of the cursor unchanged. The next **fetch** returns the next qualifying row.

- A searched or positioned **update** on an allpages-locked table can change the location of the row; for example, if it updates key columns of a clustered index. The cursor does not track the row; it remains positioned just before the next row at the original location. Positioned updates are not allowed until a subsequent **fetch** returns the next row. The updated row may be visible to the cursor a second time, if the row moves to a later position in the search order.

  Data-only-locked tables have fixed row IDs, so expanding updates or updates that affect the clustered key do not move the location of the row. The cursor remains positioned on the row, and the next **fetch** returns the next qualifying row.

The cursor positioning behavior described above is unchanged from previous releases.

Effects of Updates and Deletes on Join Cursors

When a searched or positioned **delete** is issued on the row at the cursor position of a cursor that includes a join, there can be one of two results:

- Searched deletes close the cursor implicitly. The next **fetch** returns error 582:

  ```
  Cursor 'cursor_name' was closed implicitly because
  the current cursor position was deleted due to an
  update or a delete. The cursor scan position could
  not be recovered. This happens for cursors which
  reference more than one table.
  ```

- Positioned deletes fail, with error 592:

  ```
  The DELETE WHERE CURRENT OF to the cursor
  'cursor_name' failed because the cursor is on a
  join.
  ```

  The cursor remains open, positioned at the same row.

When a searched or positioned **update** is issued on the join columns of a cursor:

- Searched updates to clustered index keys on allpages-locked tables succeed, but implicitly close the cursor, so the next fetch returns error 582. Searched updates to clustered index keys do not close cursors on data-only-locked tables.

- Positioned updates for all locking schemes, and searched updates that do not change a clustered index key on an allpages-locked table succeed and do not close the cursor.

  The cursor position depends on the type of table and whether the column has a clustered index. See "Cursor Positioning After a delete or update Command Without Joins" on page 10-11 for more information.

  Join column buffering may affect the result sets returned when join columns are updated. See "Join Cursor Processing and Data Modifications" on page 10-11 for more information.

Table 10-2 shows how **delete** and **update** commands affect join cursors.

In this table, "Left open" mans that the cursor is not closed by the update. The cursor is still positioned on the row, so positioned updates can still be made, and the next **fetch** also succeeds.

Table 10-2: Effects of deletes and join column updates in join cursors

	Version 11.9.2		11.5.x and Prior Versions
	Allpages-Locked	Data-Only-Locked	
delete commands			
Positioned direct delete	Error 592	Error 592	Error 592
Searched direct delete	Error 582	Error 582	Error 582
Searched deferred delete	Error 582	Error 582	Error 582
Searched delete affecting clustered index (deferred)	Error 582	Error 582	Error 582
update commands			
Positioned direct update	Left open	Left open	Left open
Searched direct update	Left open	Left open	Left open
Searched deferred update	Left open	Left open	Left open
Searched update affecting clustered index (deferred)	Error 582	Left open	Error 582

Effects of Join Column Buffering on Join Cursors

For cursors on queries that include joins, the join columns, search arguments and select list values for the outer table in the join order are buffered with the first **fetch**. If more than one row is returned from the inner table in the join order, subsequent fetches use the buffered

value for the outer row. This means that the behavior of applications that update join columns and search arguments through cursors can vary, depending on the join order chosen for the query. Join column buffering affects both allpages-locked tables and data-only-locked tables.

This is a change in behavior for allpages-locked tables in applications upgraded to 11.9.2. In version 11.5.x, join cursors on allpages-locked tables did not buffer join values.

Effects of Column Buffering During Cursor Scans

The results of cursor queries that perform joins may produce varying results, depending on the join order chosen for the query, if updates to the join column take place during the session. The following two tables are used to illustrate how join order can affect cursor results sets.

```
select * from dept

dept_id deptloc
------- -----------------
      1 Elm Street
      2 Acacia Drive
      3 Maple Lane
      4 Oak Avenue

select * from employee

dept_id empid
------- ---------
      1 172321176
      1 213468915
      2 341221782
      2 409567008
      2 427172319
      3 472272349
      3 486291786
```

Example of Join Column Buffering

These statements declare a cursor, open it, and fetch the first row:

```
declare j_curs cursor for
select d.dept_id, e.empid, d.deptloc
from dept d , employee e
where d.dept_id = e.dept_id
and d.dept_id = 2

open j_curs
```

```
fetch j_curs
dept_id     empid       deptloc
----------- ----------- --------------------
          2   341221782 Acacia Drive
```

If *dept* is chosen as the outer column in the join order, the value 2 for *dept_id* is buffered. The following **update** changes the *dept_id* of the join column in *dept*:

```
update dept set dept_id = 12
where current of  j_curs
```

This update changes the value stored in the table row, but does not alter the buffered value, hiding the result of this update to the base table from the cursor. The effect would be the same if the join column in *dept* were updated by a non-positioned update.

This **update** changes the *dept_id* of the *employee* row:

```
update employee set dept_id = 12
where current of j_curs
```

The next **fetch** on the table returns the second row in the result set:

```
fetch j_curs
dept_id     empid       deptloc
----------- ----------- --------------------
          2   409567008 Acacia Drive
```

All matching rows from *employee* can be fetched and updated.

If the join order for this cursor selects *employee* as the outer table, and the join column in *dept* is updated after the first **fetch**, the second **fetch** would find no matching rows. This is the sequence of steps:

1. The value 2 for *employee.dept_id* is buffered during the first fetch.
2. A searched or positioned update changes the value of *dept.dept_id* to 12.
3. At the second **fetch**, 2, the buffered value for *employee.dept_id* is used, and the inner table is scanned for matching values; since *dept.dept_id* has been changed to 12, the second **fetch** does not return a row.

The two remaining rows in *employee* with *dept_id* equal to 2 cannot be fetched by the cursor.

Example of Search Argument Buffering

Due to buffering of search arguments, the sensitivity of cursor results to updates of the search arguments on cursor select queries also

depends on join order. The following example uses the *dept* and *employee* tables as shown page 10-14. The following cursor joins on the *dept_id* columns of the table, using a search argument on the *deptloc* column:

```
declare c cursor for
select d.dept_id, empid, deptloc
from dept d , employee e
where d.dept_id = e.dept_id
and deptloc = "Acacia Drive"
```

The first **fetch** on this table returns this row:

```
dept_id   empid       deptloc
--------- ----------- --------------
        2 41221782    Acacia Drive
```

This positioned update statement changes *employee.dept_id* value to 4 for the current row; it needs to be performed for each employee in the department located at Acacia Drive:

```
update employee set dept_id = 4
where current of c
```

This positioned update changes the *dept.deptloc* value from "Acacia Drive" to "Pine Way",

```
update dept set deptloc = "Pine Way"
where current of c
```

If the **update** to *deptloc* is issued after the first **fetch**, the result set may or may not return all of the rows that need to be changed; the results depend on the join order chosen for the query:

- If *dept* is chosen as the outer table, the *dept.dept_id* and *dept.deptloc* columns, the values "2" and "Acacia Drive", are buffered. After the first **fetch**, the **update** to *deptloc* updates the base table, but not the buffered values, and subsequent **fetch** commands return additional result set rows.
- If *employee* is chosen as the outer table, only the *dept_id* qualification on the *employee* table is buffered. The **update** to the *deptloc* column changes the base table, and when the search argument qualification "Acacia Drive" is applied to the *deptloc* table at the next **fetch**, no rows are returned, even though two rows with *employee.dept_id* of 2 remain in the table.

Note that issuing the **update** to *deptloc* after all rows from the *employee* table have been fetched would accomplish both updates, without a dependency on the join order.

Effects of Select-List Buffering

Columns from the select list of the outer table in the join order are buffered when a row from the outer table is fetched. If a searched or positioned update is performed on a column in the select list, and another row is fetched from the inner table, the buffered value from the outer table appears in the cursor result row.

Recommendations

In general, applications should attempt to avoid updating join columns or columns with search clauses and other predicates to change their value when cursor scans are in progress.

- Avoid updates to the join columns of cursors or to the search arguments of join cursors whenever possible, unless you are completely sure of the join order.
- Use a cursor query that creates a worktable, and then use searched updates and deletes on the base table. When a cursor select query requires a worktable, the cursor is always insensitive to changes to the underlying table. Cursors that include **order by**, **distinct**, **group by**, or other clauses that create a worktable cannot be updated with positioned updates.
- If you are using cursors to update both join columns, consider using searched updates instead of positioned updates. Although further fetches may return buffered values instead of values that have been changed in the data rows, use of searched updates avoids the possibility of failing to **fetch** matching rows due to the choice of join order. Note searched updates to a clustered index key in an allpages-locked table implicitly close the cursor.
- Note that the ANSI SQL 92 entry-level specification does not allow updates to join columns using cursors, so the behavior is implementation specific. Applications designed to be portable across different database software must take individual implementations into account.

Update Mode and Updates Through Joins

In earlier versions of Adaptive Server and SQL Server, all updates and deletes that involve joins and subqueries are performed in deferred mode. In version 11.9.2, some of these updates and deletes can be performed in direct mode or in other update modes that are less costly than deferred mode, depending on other update mode

restrictions. The modes and restrictions that prevent direct updates are:

- deferred_varcol – the column being updated is a variable-length column. The update may be performed in either direct or deferred mode, depending on information that is available only at run time.
- deferred_index – the column being updated is part of a nonclustered, nonunique index. In this mode, Adaptive Server deletes the index entries in direct mode but inserts them in deferred mode.

In version 11.9.2, updates that involve any of the following are still performed in deferred mode:

- A self-join on the updated table
- A self-referential integrity constraint on the updated table
- A subquery correlated on the updated table

Also, in order to use direct, deferred_varcol, or deferred_index mode, the table being updated must be the outermost table in the join order, or it must be preceded in the join order by tables where only a single row qualifies. The choice of join order is made by the optimizer, based on the query cost. Although deferred updates are more expensive than direct updates, the performance penalty of deferred updates rarely outweighs the performance penalty of a poor join order.

Joins and Subqueries in Update and Delete Statements

The use of the **from** clause to perform joins in **update** and **delete** statements is a Transact-SQL® extension to ANSI SQL. Subqueries in ANSI SQL form can be used in place of joins for some updates and deletes.

This example uses the **from** syntax to perform a join:

```
update t1 set t1.c1 = t1.c1 + 50
from t1, t2
where t1.c1 = t2.c1
and t2.c2 = 1
```

The following example shows the equivalent update using a subquery:

```
update t1 set c1 = c1 + 50
where t1.c1 in (select t2.c1
                from t2
                where t2.c2 = 1)
```

The update mode that is used for the join query depends on whether the updated table is the outermost query in the join order—if it is not the outermost table, the update is performed in deferred mode. The update that uses a subquery is always performed as a direct, deferred_varcol, or deferred_index update.

For a query that uses the **from** syntax and performs a deferred update due to the join order, use **showplan** and **statistics io** to determine whether rewriting the query using a subquery can improve performance. Note that not all queries using **from** can be rewritten to use subqueries.

Deletes and Updates in Triggers vs. Referential Integrity

Triggers that join user tables with the *deleted* or *inserted* tables continue to be run in deferred mode. If you are using triggers solely to implement referential integrity, and not to cascade updates and deletes, then using declarative referential integrity in place of triggers may avoid the penalty of deferred updates in triggers.

For More Information

For more information on joins, subqueries, and update modes, see the following sections in the *Performance and Tuning Guide*:

- “Optimizing Joins” on page 8-13
- “Optimizing Subqueries” on page 8-28
- “Update Operations” on page 8-34

Queries That Use Mixed Ascending and Descending Order

Queries that use a mix of ascending and descending order in an **order by** clause do not perform a separate sort step if the index was created using the same mix of ascending and descending order as that specified in the **order by** clause, or if the index order is the reverse of the order specified in the **order by** clause. Indexes are scanned forward

or backward, following the page chain pointers at the leaf level of the index.

For example, if the *titles* table has an index on *pub_id* and *pubdate*, and *pub_id* is ascending and *pubdate* is descending, the rows are ordered on the pages, as shown in Figure 10-3. When the ascending and descending order in the query matches the index creation order, the result is a forward scan, starting at the beginning of the index or with the first qualifying rows, returning the rows in order from each page, and following the next-page pointers to read subsequent pages.

If the ordering in the query is the exact opposite of the index creation order, the result is a backward scan, starting at the last page of the index or the page containing the last qualifying row, returning rows in backward order from each page, and following previous page pointers.

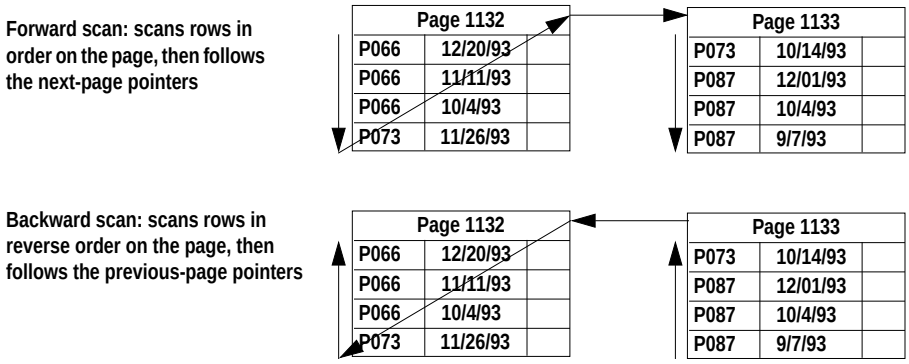

Figure 10-3: Forward and backward scans on an index

The following query using the index shown in Figure 10-3 performs a forward scan:

```
select *
from titles
order by pub_id asc, pubdate desc
```

This query using the index shown in Figure 10-3 performs a backward scan:

```
select *
from titles
order by pub_id desc, pubdate asc
```

For the following two queries on the same table, the plan requires a sort step, since the **order by** clauses do not match the ordering specified for the index:

```
select *
from titles
order by pub_id desc, pubdate desc

select *
from titles
order by pub_id asc, pubdate asc
```

Improved Performance for *order by* Queries

Index pages can be scanned by following the page chain in either direction, by following either the next page pointers on a leaf-level index page or the previous page pointers. For allpages-locked tables with a clustered index, the data pages themselves are in key order and can be scanned either forward or backward following page pointers.

A **select** query with an **order by** clause on an allpages-locked table can read the data pages directly to return rows in clustered index order, thereby avoiding a sort step. For data-only-locked tables, rows are not guaranteed to be in clustered key order, and there are no page chains on the table. For **order by** queries, the optimizer computes the cost of access via the index, and the cost of an OAM scan plus a sort.

In earlier versions of SQL Server and Adaptive Server, indexes were always created in ascending order for all keys in the index. Queries that required a mix of ascending and descending orders always:

- Selected the qualifying rows into a worktable
- Sorted the worktable to return the rows in the required order

In Adaptive Server 11.9.2, you can create indexes in descending order, or use a mix of ascending and descending order for indexes with multiple columns. Choosing an index order that matches the order in your most frequent queries can improve performance by eliminating the intermediate steps of creating the worktable and sorting it.

The worktable and sort are still required when:

- The columns listed after the **order by** clause include any column that is not part of the index.

- The columns in the **order by** clause are not a prefix subset of the keys. For columns A, B, C, D, the prefix subsets are A; A, B; A, B, C; and A, B, C, D.

  The sets of columns A, B, D and A, C, or any set without the leading column of the index are not prefix subsets and require the worktable and sort step.

- The ascending or descending ordering in the **order by** clause is not an exact match, or it is not the exact opposite, of the ascending and descending ordering for the index. For example, if the index specifies:

  ```
  (pub_id asc, pubdate desc, price asc)
  ```

  This is an exact match for the index ordering:

  ```
  order by pub_id asc, pubdate desc, price asc
  ```

  This specifies the exact opposite of the index ordering:

  ```
  order by pub_id desc, pubdate asc, price desc
  ```

  Any other combination of ascending or descending ordering requires a sort step.

Specifying Index Order When Creating Indexes

You use the **create index** command to create an index on one or more columns in a table, and specify whether the index is created in ascending or descending order for the specified column, using the **asc | desc** parameter. The syntax is:

```
create [unique] [clustered | nonclustered]
      index index_name
   on table_name (column_name [asc | desc]
      [, column_name [asc | desc]]...)
```

Use the **asc** parameter to create the index in ascending order for the column specified. Use the **desc** parameter to create the index in descending order for the column specified. If neither **asc** nor **desc** is specified for a column, the default is **asc**. For indexes with multiple keys, you can specify a mix of **asc** and **desc**. You can specify descending index order for both clustered and nonclustered indexes.

The following example creates an index named *auth_id_ix* on the *authors* table, specifying ascending order for *au_id* and descending order for *city*.

```
create index auth_id_ix
on authors (au_id asc, city desc)
```

The **asc** and **desc** specifications can also be included when you are creating constraints that require indexes. This **alter table** command adds a constraint on the sales table:

```
alter table sales
    add constraint stor_ord primary key
    (stor_id asc, ord_num desc)
```

Changes to *showplan* for Descending Scans

The output from **showplan** has changed slightly to support these changes. The changes are:

- **showplan** reports the scan direction messages in the FROM TABLE output as follows:
  - "Ascending scan" is now "Forward scan"
  - "Descending scan" is now "Backward scan"
- **showplan** shows the ascending or descending ordering requested in the query after the column name, for example:

```
Keys are:
            pub_id ASC
            pubdate DESC
```

If the sort must be performed, **showplan** output includes an extra step for creating the worktable and messages indicating the sort process. See "Worktable Message for order by" on page 9-21 and "Sorting Messages" on page 9-22 of the *Performance and Tuning Guide* for more information.

Choosing Index Ordering

Scans that include **order by** can avoid the sort step when the physical ordering of the keys is either the same as or the opposite of, the direction of ordering specified for the query.

If your goal is to reduce the total amount of work performed by the server, choose an order that matches the most frequent ordering that is used or the ordering used in the largest result sets. If your goal is to improve the response time of critical queries that use **order by**, eliminating the sort step can not only reduce the total time taken to execute the query, but also reduce the time taken to return the first row.

Query Plan Recompilation

In version 11.9.2, stored procedures are recompiled for the following additional reasons:

- When the locking scheme for a table changes
- When the read-only status for a database is changed with **sp_dboption**
- When the isolation level changes between the time the query is compiled and the time it is executed
- When **reorg rebuild** is run on a table

11 Changes to *showplan* and *dbcc traceon(302)* Output

This chapter contains the following sections:

New and Changed *showplan* Messages

The following describes changes to **showplan** messages:

- The message "Using I/O Size N Kbytes" is now displayed for both index and data pages. The new messages are:

  ```
  - Using I/O Size N Kbytes for index leaf pages.
  - Using I/O Size N Kbytes for data pages.
  ```

- The message "With [LRU | MRU] Buffer Replacement Strategy" is now displayed for both index and data pages. The new messages are:

  ```
  - With [LRU|MRU] Buffer Replacement Strategy for
    data pages.
  - With [LRU|MRU] Buffer Replacement Strategy for
    index leaf pages.
  ```

- "Ascending scan." has been changed to "Forward scan."
- "Descending scan." has been changed to "Backward scan."
- The ascending or descending ordering is now shown after the column name, for example:

  ```
  Keys are:
              pub_id ASC
              pubdate DESC
  ```

If a query is optimized using simulated statistics, the following message is displayed:

```
Optimized using simulated statistics.
```

For a comparison of **showplan** messages for **select** statements related to forward and backward scans, see "Improved Performance for order by Queries" on page 10-21.

New and Changed *dbcc traceon(302)* Messages

The output for **dbcc traceon(302)** has been improved and expanded in version 11.9.2. Messages have been made less cryptic and messages have been added to support new features.

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, and 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, which is 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*

Figure 11-1 shows the ordering and basic structure of the search and join blocks in **dbcc traceon(302)** output.

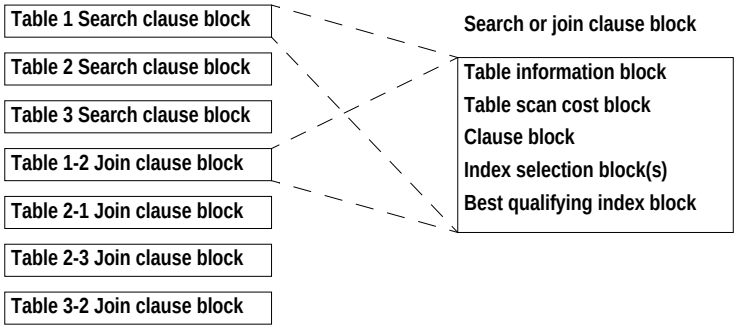

Figure 11-1: Structure of blocks of output in dbcc traceon(302)

Additional Blocks

Some queries generate additional blocks 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 11-8 for more information.
- Queries that use a worktable contain a block to display the cost of using a worktable.
- 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.
  ```

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. **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 "Choosing the Query Plan Based on Costs" on page 11-16 for information on **dbcc traceon(310)**.

The 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 row gives 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 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 14, "Parallel Query Optimization," in the *Performance and Tuning Guide* for more information on parallel query optimization.

The Base Cost Block

Before starting to analyze query clauses and useful indexes for a table, the optimizer determines the cost of a table scan. 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
```

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.
```

By default, when a data-only-locked table has 15 or fewer pages, the optimizer chooses index access even when a table scan require fewer I/Os, if any index on the table matches an optimizable search argument in the query. Index access provides improvement in concurrency, with a possible slight increase in I/O. For more information, see "Concurrency Optimization for Small Tables" on page 10-7.

The 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. If there are no search clauses for a table, **dbcc traceon(302)** prints:

```
No Search clauses for table
```

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.

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 which uses the **getdate** function:

```
where pubdate < getdate()
```

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, all of 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
```

Table 11-1 shows the codes that were used in pre-11.9.2 versions of **dbcc traceon(302)** and the equivalent operators in 11.9.2.

Table 11-1: Operators in dbcc traceon(302) output

Previous Code	Operator	Comparison in the Query Clause
EQ	=	Equality comparison
LT	<	Less than comparison
LE	<=	Less than or equal to comparison
GT	>	Greater than comparison
GE	>=	Greater than or equal to comparison
NE	!=	Not equals (!=) comparison
ISNULL	**is null**	**is null** comparison
ISNOTNULL	**is not null**	**is not null** comparison

Sort Avert Messages

If the query includes an **order by** clause, additional messages are displayed while the optimizer analyzes the best index to use to avoid incurring sort costs. 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 clause block, 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.
```

The Column Block

This section prints the selectivity of each optimizable search argument or join clause. Selectivity is used to estimate the number of matching values 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. If the unknown value is a search argument on an indexed column, the total density for the column is used as the default selectivity. This message indicates an unknown value for a column where statistics are available:

```
SARG is a local variable or the result of a function or an
expression, using the join density to estimate selectivity.
```

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.
```

The default values depend on the operator in the clause:

- Equality – 10 percent
- In between – 25 percent
- Open-ended range – 33 percent

You may be able to improve optimization for queries where these messages appear frequently, by creating statistics on the columns. See "Creating and Updating Column Statistics" on page 9-4.

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, 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. Here are sample messages:

```
User forces index 2 (index name = title_id_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 updated using **optdiag**, **dbcc traceon(302)** prints the message:

```
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 earlier version of the server, **dbcc traceon(302)** prints the message:

```
Statistics for this column were obtained from
upgrade.
```

If this message appears, you should run **update statistics** for the table or index.

The 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 Clustering Ratio 0.990535,
   Index Page Clustering Ratio 0.538462,
   Data Page Clustering Ratio 0.933579
```

Scan and Filter Selectivity Values

This block includes two selectivity values, a scan value and a filter value. In the example above, these values are the same (0.000936). For queries that specify search arguments on multiple keys in an index, these values can differ. The two selectivities are used for different purposes, as shown in Table 11-2.

Table 11-2: Scan and filter selectivity

	Scan Selectivity	Filter Selectivity
Used to estimate:	Number of index rows and 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

How Scan and Filter Selectivity Can Differ

This statement creates a composite index on *titles*:

```
create index composite_ix
on titles (pub_id, pubdate, total_sales)
```

Both of the following clauses can be used to position the start of the search and to limit the end point, since both of the leading columns are specified:

```
where pub_id = "P099"
where pub_id = "P099" and pubdate = "10/1/93"
```

The first example requires reading all the index pages where *pub_id* equals”P099”. The second example reads only the pages where both conditions are true. It needs to read fewer index pages, and returns fewer rows, so it needs to read fewer data pages.

In the following example, the *pub_id* clause can be used to position the scan and limit the end point, as in the queries above. The *total_sales* clause limits the number of rows from the index that match the query’s qualifications, thereby limiting the number of data pages that need to be accessed:

```
where pub_id = "P099" and total_sales > 10000
```

This query needs to read all the index pages where *pub_id* equals “P099”. As it examines the values on the leaf pages of the index, it applies the *total_sales* search criteria to the index values to determine which data pages it needs to access. The filter selectivity estimates

the percentage of rows from this step that qualify and how many data pages need to be accessed.

In the **dbcc traceon(302)** output below, the selectivity for the *total_sales* column uses the default value, 0.33, for an open range scan. When combined with the selectivity of 0.003223 for *pub_id*, it yields the filter selectivity of 0.001063 for *composite_ix*:

```
Selecting best index for the SEARCH CLAUSE:
        titles.total_sales > 10000
        titles.pub_id = 'P099'


Estimated selectivity for pub_id,
   selectivity = 0.003223, upper limit = 0.066600.

No statistics available for total_sales,
using the default range selectivity to estimate selectivity.

Estimated selectivity for total_sales,
   selectivity = 0.330000.

Estimating selectivity of index 'composite_ix', indid 6
   scan selectivity 0.003223, filter selectivity 0.001063
   5 rows, 8 pages, index height 1,
   Data Row Cluster Ratio 0.001404,
   Index Page Cluster Ratio 1.000000,
   Data Page Cluster Ratio 0.011014
```

Rows, Pages, and Index Height

This row of output prints the estimated number of rows, the number of data and index pages that need to be read, and the height of the index.

Cluster Ratios

The three cluster ratio values for the index are included; these are the same values as those displayed by **optdiag**. See "Index Cluster Ratios" on page 8-12 for more information.

Covering Indexes

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.

The Best Access Block

The final section for each index selection block shows the best qualifying index for the clauses examined in the block. The block does not necessarily indicate the index to be used for the query plan.

```
The best qualifying index is 'au_names_ix' (indid 2)
   costing 7 pages,
   with an estimate of 5 rows to be returned per scan of the table,
  using no index prefetch (size 2K I/O) on leaf pages, no data
prefetch (size 2K I/O),
   on index cache 'pubtune_cache' (cacheid 1) with LRU replacement
   on data cache 'pubtune_cache' (cacheid 1) with LRU replacement
Search argument selectivity is 0.000937.
```

This block repeats much of the information shown in the index selection block for the index chosen as the best. The only additional information is the I/O size, cache, and buffer replacement strategy information for both index pages and data pages.

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 in the join order, the total cost of accessing the table will be 7 pages, and it will return 5 rows. If this query is an inner table of a join, and the outer table returns 3 rows, this table will be accessed 3 times, with a cost of 7 pages each time, and each access is estimated to return 5 rows.

Choosing the Query Plan Based on 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 only concerned 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. The **dbcc traceon(310)** option provides information about the join order selection step.

dbcc traceon(310) prints the first plan that the optimizer costs, and then each cheaper plan, with the heading "NEW PLAN". If you want to see all of the plans, use **dbcc traceon(317)**. It prints each plan considered, with the heading "WORK PLAN". (This may produce a lot of output, especially for queries with many tables, many indexes, and numerous query clauses.)

The **dbcc traceon(310)** or **(317)** facility 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.

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. The same format and values are used in the "NEW PLAN" and "WORK PLAN" output.

```
FINAL PLAN (total cost = 39427):

varno=0 (titles) indexid=3 (type_price)
path=0xdc376980 pathtype=pll-sarg-nc
method=NESTED ITERATION
numthreads = 3
outerrows=1 rows=138 joinsel=1.000000 cpages=134 index_prefetch=NO
index_iosize=2 index_bufreplace=LRU data_prefetch=NO
data_iosize=2 data_bufreplace=LRU lio=49 pio=15 corder=3
```

```
varno=2 (titleauthor) indexid=0 ()
path=0xdc399800 pathtype=join
method=NESTED ITERATION
numthreads = 1
outerrows=138 rows=172 joinsel=0.000200 cpages=106 data_prefetch=NO
data_iosize=2 data_bufreplace=LRU lio=4947 pio=36 corder=0
jnvar=0 refcost=51203 refpages=2 reftotpages=53 ordercol[0]=2
ordercol[1]=1

varno=1 (authors) indexid=0 ()
path=0xdc399a88 pathtype=join
method=NESTED ITERATION
numthreads = 1
outerrows=172 rows=43 joinsel=0.000200 cpages=223 data_prefetch=NO
data_iosize=2 data_bufreplace=LRU lio=13008 pio=75 corder=0
jnvar=2 refcost=42020 refpages=2 reftotpages=32 ordercol[0]=1
ordercol[1]=1
```

Table 11-3 shows the meaning of the values in the output.

Table 11-3: 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 also provided.
indexid	The index ID, followed by the index name
pathtype	The access method for this table. See Table 11-4.
method	One of: “NESTED ITERATION”, “REFORMATTING”, “REFORMATTING with Unique Reformatting”, or “OR OPTIMIZATION”
numthreads	Number of worker processes to be used for the table
outerrows	Number of rows that qualify in the outer tables in the query or 1, for the first table in the join order
rows	Number of rows estimated to qualify in this table. For a parallel query, this is the maximum number of rows per worker process.
joinsel	The join selectivity
cpages	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)
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)

Table 11-3: dbcc traceon(310) output (continued)

Label	Information Provided
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)
lio	The estimated number of logical I/Os. For a parallel query, this is the maximum number of logical I/Os per worker process.
pio	The estimated number of physical I/Os. For a parallel query, this is the maximum number of physical I/Os per worker process.
corder	The order of the column used
jnvar	The *varno* of the previous table in the join order, for second and subsequent tables in the join order
refcost	The total cost of reformatting, estimated as: `2 * logical I/Os + 18 * physical I/Os` Included for the second and subsequent tables in the join order
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
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 11-4 shows the access methods that correspond to the *pathtype* information in the **dbcc traceon(310)** output.

Table 11-4: 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 nonclustered index scan on a search clause
pll-join-nc	Parallel nonclustered index 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

New and Changed Syntax and Commands

12 New and Changed Transact-SQL Commands

This chapter documents the following new and changed Transact-SQL commands:

➤ ***Note***

In this chapter, the syntax for new commands is shown entirely in boldface type. New clauses for existing commands are shown in boldface type, with the remainder of the syntax shown in regular type. Only new parameters and functionality are covered here. See the *Adaptive Server Reference Manual* for more information on these commands.

New Reserved Words

The new reserved words in version 11.9.2 are:

- **exp_row_size**
- **reservepagegap**
- **lock**
- **readpast**
- **reorg**

The following words were reserved words in 11.9 and 11.5.x versions of Adaptive Server. These words are no longer treated as reserved words in 11.9.2:

- **activation**
- **consumers**
- **force_ddl**
- **membership**
- **passwd**
- **session**
- **then**

alter table

New Functionality

Changes the locking scheme for an existing table; specifies ascending or descending index order when **alter table** is used to create referential integrity constraints that are based on indexes; specifies the ratio of filled pages to empty pages, to reduce storage fragmentation.

Syntax

```
alter table [database.[owner].]table_name
{ add column_name datatype
    [default {constant_expression | user | null}]
    {identity | null}
    [ [constraint constraint_name]
       { { unique | primary key }
           [clustered | nonclustered] [asc | desc]
           [with { { fillfactor = pct
                   | max_rows_per_page = num_rows }
                   , reservepagegap = num_pages }]
            [on segment_name]
       | references [[database.]owner.]ref_table
            [(ref_column)]
       | check (search_condition) ] ... }
    [, next_column]...
| add { [constraint constraint_name]
      { {unique | primary key}
           [clustered | nonclustered]
           (column_name [asc | desc]
                [, column_name [asc | desc]...])
            [with { { fillfactor = pct
                    | max_rows_per_page = num_rows}
                    , reservepagegap = num_pages}]
           [on segment_name]
      | foreign key (column_name [,column_name...])
            references [[database.]owner.]ref_table
             [(ref_column [, ref_column...])]
      | check (search_condition)} }
| drop constraint constraint_name
| replace column_name
    default {constant_expression | user | null}
| partition number_of_partitions
| unpartition
| lock {allpages | datarows | datapages } }
```

New Parameters and Keywords

asc | desc – specifies whether the index is to be created in ascending (**asc**) or descending (**desc**) order. The default is ascending order.

reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations for the index created by the constraint. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value, 0, leaves no empty pages.

lock datarows | datapages | allpages – changes the locking scheme to be used for the table.

Examples

1.
```
alter table titles lock datarows
```

   Changes the locking scheme for the *titles* table to datarows locking.

2.
```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
with reservepagegap = 16
```

   Creates an index on *authors*; the index has a **reservepagegap** value of 16, leaving 1 empty page in the index for each 15 allocated pages.

Comments

Changing Locking Schemes

- **alter table** supports changing from any locking scheme to any other locking scheme. You can change:
  - From **allpages** to **datapages** or vice versa
  - From **allpages** to **datarows** or vice versa
  - From **datapages** to **datarows** or vice versa
- Before you change from allpages locking to a data-only locking scheme, or vice versa, you must first use **sp_dboption** to set the database option **select into/bulkcopy/pllsort** to true and then run **checkpoint** in the database if any of the tables are partitioned and the sorts for the indexes require a parallel sort.
- After changing the locking scheme from allpages-locking to data-only locking or vice versa, the use of the **dump transaction** command

to back up the transaction log is prohibited; you must first perform a full database dump.

- When you use **alter table...lock** to change the locking scheme for a table from allpages locking to data-only locking or vice versa, Adaptive Server makes a copy of the table's data pages. There must be enough room on the segment where the table resides for a complete copy of the data pages. There must be space on the segment where the indexes reside to rebuild the indexes.

  Clustered indexes for data-only-locked tables have a leaf level above the data pages. If you are altering a table with a clustered index from allpages-locking to a data-only-locking, the resulting clustered index requires more space. The additional space required depends on the size of the index keys.

  Use **sp_spaceused** to determine how much space is currently occupied by the table, and use **sp_helpsegment** to see the space available to store the table.

- When you change the locking scheme for a table from allpages locking to datapages locking or vice versa, the space management properties are applied to the tables, as the data pages are copied, and to the indexes, as they are re-created. When you change from one data-only locking scheme to another, the data pages are not copied, and the space management properties are not applied.

- When you change the locking scheme for a table, the **alter table...lock** command acquires an exclusive lock on the table until the command completes.

- When you use **alter table...lock** to change from datapages locking to datarows locking, the command does not copy data pages or rebuild indexes. It only updates system tables.

- Changing the locking scheme while other users are active on the system may have the following effects on user activity:

  - Query plans in the procedure cache that access the table will be recompiled the next time they are run.

  - Active multi-statement procedures that use the table are recompiled before continuing with the next step.

  - Ad hoc batch transactions that use the table are terminated.

- For more information on changing the locking scheme for existing tables, see Chapter 3, "Creating and Managing Data-Only-Locked Tables."

◆ ***WARNING!***

Changing the locking scheme for a table while a bulk copy operation is active can cause table corruption. Bulk copy operates by first obtaining information about the table and does not hold a lock between the time it reads the table information and the time it starts sending rows, leaving a small window of time for an alter table...lock command to start.

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing. See "Improved Performance for order by Queries" on page 10-21 for more information.

Setting Space Management Properties for Indexes

- The space management properties **fillfactor**, **max_rows_per_page**, and **reservepagegap** in the **alter table** statement apply to indexes that are created for **primary key** or **unique** constraints. The space management properties affect the data pages of the table if the constraint creates a clustered index on an allpages-locked table.
- Use **sp_chgattribute to** change **max_rows_per_page** or **reservepagegap** for a table or an index, or to store **fillfactor** values.
- Space management properties for indexes are applied:
  - When indexes are re-created as a result of an **alter table** command that changes the locking scheme for a table from allpages locking to data-only locking or vice versa. See "Changing Locking Schemes" on page 12-4 for more information.
  - When indexes are automatically rebuilt as part of a **reorg rebuild** command.
- To see the space management properties currently in effect for a table, use **sp_help**. To see the space management properties currently in effect for an index, use **sp_helpindex**.

- The space management properties **fillfactor**, **max_rows_per_page**, and **reservepagegap** help manage space usage for tables and indexes in the following ways:
  - **fillfactor** leaves extra space on pages when indexes are created, but the **fillfactor** is not maintained over time. It applies to all locking schemes.
  - **max_rows_per_page** limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables.
  - **reservepagegap** specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to all locking schemes.

  Space management properties can be stored for tables and indexes so that they are applied during alter table and reorg rebuild commands. See Chapter 4, "Setting Space Management Properties," for more information.
- Table 12-1 shows the valid combinations of space management properties and locking schemes. If an **alter table** command changes the table so that the combination is not compatible, the values stored in the stored in system tables remain there, but are not applied during operations on the table. If the locking scheme for a table changes so that the properties become valid, then they are used.

Table 12-1: Space management properties and locking schemes

Parameter	*allpages*	*datapages*	*datarows*
max_rows_per_page	Yes	No	No
reservepagegap	Yes	Yes	Yes
fillfactor	Yes	Yes	Yes
exp_row_size	No	Yes	Yes

- Table 12-2 shows the default values and the effects of using the default values for the space management properties.

Table 12-2: Defaults and effects of space management properties

Parameter	Default	Effect of Using the Default
max_rows_per_page	0	Fits as many rows as possible on the page, up to a maximum of 255

Table 12-2: Defaults and effects of space management properties

Parameter	Default	Effect of Using the Default
reservepagegap	0	Leaves no gaps
fillfactor	0	Fully packs leaf pages

Conversion of *max_rows_per_page* to *exp_row_size*

- If a table has **max_rows_per_page** set, and the table is converted from allpages locking to data-only locking, the value is converted to an **exp_row_size** value before the **alter table...lock** command copies the table to its new location. The **exp_row_size** is enforced during the copy. Table 12-3 shows how the values are converted.

Table 12-3: Converting max_rows_per_page to exp_row_size

If *max_rows_per_page* is set to	Set *exp_row_size* to
0	Percentage value set by **default exp_row_size percent**
255	1, that is, fully packed pages
1–254	The smaller of: • maximum row size • 2002 / **max_rows_per_page** value

Using *reservepagegap*

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The **reservepagegap** keyword causes these commands to leave empty pages so that future page allocations take place close to the page that is being split or to the page from which a row is being forwarded.
- The **reservepagegap** value for a table is stored in *sysindexes*, and is applied when the locking scheme for a table is changed from allpages locking to data-only locking or vice versa. To change the stored value, use the system procedure **sp_chgattribute** before running **alter table**.
- **reservepagegap** specified with the **clustered** keyword on an allpages-locked table overwrites any value previously specified with **create table** or **alter table**.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

alter table permission defaults to the table owner. It cannot be transferred except to the Database Owner, who can impersonate the table owner by running the **setuser** command. A System Administrator can also alter user tables.

See Also

Commands	**create index, create table**
System procedures	**sp_chgattribute, sp_help**

create existing table

(Component Integration Services only)

New Functionality

Allows you to map the proxy table to a table, view, or procedure at a remote location.

Syntax

```
create existing table table_name (column_list)
[ on segment_name ]
[ [ external {table | procedure} ] at pathname ]
```

New Parameters and Keywords

external – specifies that the object is a remote object.

table – specifies that the remote object is a table or a view. The default is **external table**.

procedure – specifies that the remote object is a stored procedure.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object

where:

- *server_name* (required) is the name of the server that contains the remote object
- *dbname* (optional) is the name of the database managed by the remote server that contains this object
- *owner* (optional) is the name of the remote server user that owns the remote object
- *object* (required) is the name of the remote table, view, or procedure

Examples

```
1. create existing table blurbs
   (author_id  id      not null,
   copy        text    not null)
   at "SERVER_A.db1.joe.blurbs"
```

Creates a proxy table named *blurbs* for the *blurbs* table at the remote server SERVER_A.

```
2. create existing table rpc1
   (column_1   int,
   column_2    int)
   external procedure
   at "SERVER_A.db1.joe.p1"
```

Creates a proxy table named *rpc1* for the remote procedure named *p1*.

Comments

- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.
- Component Integration Services inserts or updates a record in the *systabstats* catalog for each index of the remote table. Since detailed structural statistics are irrelevant for remote indexes, only a minimum number of columns are set in the *systabstats* record—*id*, *indid*, and *rowcnt*.

Changes by Server Class

- All server classes now allow you to specify fewer columns than there are in the table on the remote server.
- All server classes now match the columns by name. Some server classes previously matched columns by column ID.
- All server classes now allow the column type to be any datatype that can be converted to and from the datatype of the column in the remote table.

Remote Procedures

- When the proxy table is a procedure-type table, you must provide a column list that matches the description of the remote procedure's result set. **create existing table** does **not** verify the accuracy of this column list.
- No indexes are created for procedures.
- Component Integration Services treats the result set of a remote procedure as a virtual table that can be sorted, joined with other tables, or inserted into another table using **insert** or **select**. However, a procedure type table is considered read-only, which means you cannot issue the following commands against the table:
  - **delete**

  - **update**
  - **insert**
  - **create index**
  - **truncate table**
  - **alter table**

- Begin the column name with an underscore (_) to specify that the column is not part of the remote procedure's result set. These columns are referred to as parameter columns. For example:

```
create existing table rpc1
(
    a       int,
    b       int,
    c       int,
    _p1     int null,
    _p2     int null
)
external procedure
at "SYBASE.sybsystemprocs.dbo.myproc"
```

  In this example, the parameter columns *_p1* and *_p2* are input parameters. They are not expected in the result set, but can be referenced in the query:

```
select a, b, c from t1
where _p1 = 10 and _p2 = 20
```

  CIS passes the search arguments to the remote procedure as parameters, using the names *@p1* and *@p2*.

- Parameter column definitions in a **create existing table** statement must follow these rules:
  - Parameter column definitions must allow a null value.
  - Parameter columns cannot precede regular result columns—they must appear at the end of the column list.
- If a parameter column is included in a **select** list **and** is passed to the remote procedure as a parameter, the return value is assigned by the **where** clause.
- If a parameter column is included in a **select** list, but does not appear in the **where** clause or cannot be passed to the remote procedure as a parameter, its value is NULL.
- A parameter column can be passed to a remote procedure as a parameter if the Adaptive Server query processor considers it a searchable argument. A parameter column is considered a

searchable argument if it is not included in any **or** predicates. For example, the **or** predicate in the second line of the following query prevents the parameter columns from being used as parameters:

```
select a, b, c from t1
where _p1 = 10 or _p2 = 20
```

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create existing table permission defaults to the table owner and is not transferable.

See Also

Commands	**create table**, **create proxy_table**

create index

New Functionality

Creates an index in ascending or descending order for each column; allows up to 31 columns per index; leaves a specified number of unused pages during index creation; allows specification of the number of steps in the distribution histogram for the index.

Syntax

```
create [unique] [clustered | nonclustered]
    index index_name
on [[database.]owner.]table_name
    (column_name [asc | desc]
    [, column_name [asc | desc]]...)
[with {
    {fillfactor = pct| max_rows_per_page=num_rows},
    reservepagegap = num_pages,
    consumers = x, ignore_dup_key, sorted_data,
    [ignore_dup_row | allow_dup_row],
    , statistics using num_steps values  } ]
[on segment_name]
```

New Keywords and Options

asc | desc – specifies whether the index is to be created in ascending or descending order for the column specified. The default is ascending order.

with reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the index. Valid values are 0–255. The default is 0. See "Space Management Properties" on page 12-16 for more information on setting **reservepagegap**.

with statistics using *num_steps* **values** – specifies the number of steps to generate for the histogram used to optimize queries. If this clause is omitted:

- The default value is 20, if no histogram is currently stored for the leading index column,
- The current number of steps is used, if a histogram for the leading column of the index column already exists.

If you specify 0 for *num_steps*, the index is re-created, but the statistics for the index are not overwritten in the system tables.

Examples

1.
```
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
```

   Creates an index with ascending ordering on *pub_id* and descending order on *pubdate*.

2.
```
create index title_id_ix
    on titles (title_id)
    with reservepagegap = 40,
    statistics using 50 values
```

   Creates an index on *title_id*, using 50 histogram steps for optimizer statistics and leaving 1 empty page out of every 40 pages in the index.

Comments

- **create index** allows specifying up to 31 columns (formerly 16) for the index key. The maximum total number of bytes is 600.
- When using parallel sort for data-only-locked tables, the number of worker processes must be configured to equal or exceed the number of partitions, even for empty tables. The database option **select into/bulkcopy/pllsort** must also be enabled.
- The maximum number of indexes allowed on a data-only-locked table with a clustered index is 249. A table can have 1 clustered index and 248 nonclustered indexes.

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sorting order for the index keys. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing. See “Improved Performance for order by Queries” on page 10-21 for more information.

Specifying the Number of Histogram Steps

- Use the **with statistics** clause to specify the number of steps for a histogram for the leading column of an index. Histograms are used during query optimization to determine the number of rows

that match search arguments for a column. See "Understanding Histogram Output" on page 8-19 for more information.

- To re-create an index without updating the values in *sysstatistics* for a column, use 0 for the number of steps. This avoids overwriting statistics that have been changed with **optdiag**.

Space Management Properties

- **fillfactor, max_rows_per_page**, and **reservepagegap** help manage space on index pages in different ways:
  - **fillfactor** applies to indexes for all locking schemes. For clustered indexes on allpages-locked tables, it affects the data pages of the table. On all other indexes, it affects the leaf level of the index.
  - **max_rows_per_page** applies only to index pages of allpages-locked tables.
  - **reservepagegap** applies to tables and indexes for all locking schemes.
- **reservepagegap** affects space usage in indexes:
  - At the time the index is created
  - When **reorg** commands on indexes are executed
  - When nonclustered indexes are rebuilt after creating a clustered index
- When a **reservepagegap** value is specified in a **create clustered index** command, it applies:
  - To the data and index pages of allpages-locked tables
  - To only the index pages of data-only-locked tables
- The *num_pages* value specifies a ratio of filled pages to empty pages on the leaf level of the index so that indexes can allocate space close to existing pages, as new space is required. For example, a **reservepagegap** of 10 leaves 1 empty page for each 9 used pages.
- **reservepagegap** specified along with **create clustered index** on an allpages-locked table overwrites any value previously specified with **create table** or **alter table**.
- You can change the space management properties for an index with **sp_chgattribute**. Changing properties with **sp_chgattribute** does not immediately affect storage for indexes on the table. Future

large scale allocations, such as running **reorg rebuild**, use the **sp_chgattribute** value.

- The **fillfactor** value set by **sp_chgattribute** is stored in the *fill_factor* column in *sysindexes*. The **fillfactor** is applied when an index is re-created as a result of an **alter table...lock** command or a **reorg rebuild** command.
- See Chapter 4, "Setting Space Management Properties," for more information.

Index Options and Locking Modes

- Table 12-4 shows the index options supported for allpages-locked and data-only-locked tables. On data-only-locked tables, the **ignore_dup_row** and **allow_dup_row** options are enforced during **create index**, but are not enforced during **insert** and **update** operations. Data-only-locked tables always allow the insertion of duplicate rows.

Table 12-4: create index options supported for locking schemes

Index Type	Allpages-Locked Table	Data-Only-Locked Table	
		During Index Creation	During Inserts
Clustered	**allow_dup_row** **ignore_dup_row**	**allow_dup_row** **ignore_dup_row**	**allow_dup_row**
Unique clustered	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**
Nonclustered	None	None	None
Unique nonclustered	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**

Table 12-5 shows the behavior of commands that attempt to insert duplicate rows into tables with clustered indexes, and when the clustered indexes are dropped and re-created.

Table 12-5: Enforcement and errors for duplicate row options

Options	Allpages-Locked Table	Data-Only-Locked Table
No options specified	Insert fails with error message 2615. Re-creating the index succeeds.	Insert succeeds. Re-creating the index fails with error message 1508.
allow_dup_row	Insert and re-creating the index succeed.	Insert and re-creating the index succeed.
ignore_dup_row	Insert fails with "Duplicate row was ignored" message. Re-creating the index succeeds.	Insert succeeds. Re-creating the index deletes duplicate rows.

Using the *sorted_data* Option on Data-Only-Locked Tables

- The **sorted_data** option to **create index** can be used only immediately following a bulk copy operation into an empty table. Once data modifications to that table cause additional page allocations, the **sorted_data** option cannot be used.
- Specifying different values for space management properties may override the sort suppression functionality of the **sorted_data**. See "reservepagegap and sorted_data Options to create index" on page 4-13 and "Use of the sorted_data and fillfactor Options" on page 4-20 for more information.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create index permission defaults to the table owner and is not transferable.

See Also

Commands	**alter table**
System procedures	**sp_chgattribute, sp_helpindex**

create proxy_table

(Component Integration Services only)

Function

Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

Syntax

```
create proxy_table table_name
   [ external table ]
   at pathname
```

Keywords and Options

table_name – specifies the local proxy table name to be used by subsequent statements. *table_name* takes the form:

dbname.owner.object

where *dbname* and *owner* are optional and represent the local database and owner name. If *dbname* is not specified, the table is created in the current database; if *owner* is not specified, the table is owned by the current user. If either *dbname* or *owner* is specified, the entire *table_name* must be enclosed in quotes. If only *dbname* is present, a placeholder is required for *owner*.

external table – specifies that the object is a remote table or view. **external table** is the default, so this clause is optional.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object

where:

- *server_name* (required) is the name of the server that contains the remote object
- *dbname* (optional) is the name of the database managed by the remote server that contains this object
- *owner* (optional) is the name of the remote server user that owns the remote object
- *object* (required) is the name of the remote table or view

Examples

1. **create proxy_table t1**
   at "SERVER_A.db1.joe.t1"

   Creates a proxy table named *t1* that is mapped to the remote table *t1*. CIS derives the column list from the remote table.

Comments

- **create proxy_table** is a variant of the **create existing table** command. You use **create proxy_table** to create a proxy table, but (unlike **create existing table**) you do not specify a column list. CIS derives the column list from the metadata it obtains from the remote table.
- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.
- If the remote server object does not exist, the command is rejected with an error message.
- If the object exists, the local system tables are updated. Every column is used. Columns and their attributes are obtained for the table or view.
- CIS automatically converts the datatype of the column into an Adaptive Server datatype. If the conversion cannot be made, the **create proxy_table** command does not allow the table to be defined.
- Index information from the remote server table is extracted and used to create rows for the system table *sysindexes*. This defines indexes and keys in Adaptive Server terms and enables the query optimizer to consider any indexes that may exist on the table.
- After defining the proxy table, issue an **update statistics** command for the table. This allows the query optimizer to make intelligent choices regarding join order.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

create proxy_table permission defaults to the table owner and is not transferable.

See Also

Commands	**create existing table**, **create table**

create table

New Functionality

Specifies a locking scheme for the table being created; specifies ascending or descending index order when creating referential integrity constraints that depend on indexes; specifies the expected row size, to reduce row forwarding; specifies a ratio of empty pages to be left for each filled page; allows you to map the table to a table, view, or procedure at a remote location.

Syntax

```
create table [database.[owner].]table_name
(column_name datatype
    [default {constant_expression | user | null}]
    {[{identity | null | not null}]
    | [[constraint constraint_name]
        {{unique | primary key}
       [clustered | nonclustered] [asc | desc]
       [with { { fillfactor = pct
                   | max_rows_per_page = num_rows }
                   , reservepagegap = num_pages }]
        [on segment_name]
        | references [[database.]owner.]ref_table
            [(ref_column)]
        | check (search_condition)}]}...
    | [constraint constraint_name]
    {{unique | primary key}
        [clustered | nonclustered]
        (column_name [asc | desc]
            [{, column_name [asc | desc]}...])
       [with { {fillfactor = pct
              | max_rows_per_page = num_rows },
            reservepagegap = num_pages } ]
         [on segment_name]
    |foreign key (column_name [{, column_name}...])
        references [[database.]owner.]ref_table
            [(ref_column [{, ref_column}...])]
     | check (search_condition) ... }
[{, {next_column | next_constraint}}...])
[lock {datarows | datapages | allpages }]
[with { max_rows_per_page = num_rows ,
        exp_row_size = num_bytes ,
        reservepagegap = num_pages } ]
[on segment_name]
[ [ external table ] at pathname ]
```

New Keywords and Options

asc | desc – specifies whether the index created for a constraint is to be created in ascending or descending order for each column. The default is ascending order.

lock datarows | datapages | allpages – specifies the locking scheme to be used for the table. The default is the server-wide setting for the configuration parameter **lock scheme**.

exp_row_size = *num_bytes* – specifies the expected row size; applies only to **datarows** and **datapages** locking schemes, and only to tables with variable-length rows. Valid values are 0, 1, and any value between the minimum and maximum row length for the table. The default value is 0, which means a server-wide setting is applied.

reservepagegap = *num_pages* – specifies the ratio of filled pages to empty pages that are to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

external table – specifies that the object is a remote table or view. **external table** is the default, so specifying this is optional.

at *pathname* – specifies the location of the remote object. *pathname* takes the form:

server_name.dbname.owner.object;aux1.aux2

where:

- *server_name* (required) is the name of the server that contains the remote object.
- *dbname* (optional) is the name of the database managed by the remote server that contains this object.
- *owner* (optional) is the name of the remote server user that owns the remote object.
- *object* (required) is the name of the remote table or view.
- *aux1.aux2* (optional) is a string of characters that is passed to the remote server during a **create table** or **create index** command. This string is used only if the server is class *db2*. *aux1* is the DB2 database in which to place the table, and *aux2* is the DB2 tablespace in which to place the table.

Examples

1.
```
create table new_titles (
        title_id    tid,
        title       varchar(80) not null,
        type        char(12) ,
        pub_id      char(4) null,
        price       money null,
        advance     money null,
        total_sales int null,
        notes       varchar(200) null,
        pubdate     datetime,
        contract    bit    )
   lock datapages
   with exp_row_size = 200
```

   Specifies the **datapages** locking scheme for the *new_titles* table and an expected row size of 200.

2.
```
create table new_publishers (
pub_id          char(4) not null,
pub_name        varchar(40) null,
city            varchar(20) null,
state           char(2) null )
lock datarows
with reservepagegap = 16
```

   Specifies the **datarows** locking scheme and sets a **reservepagegap** value of 16 so that extent I/O operations leave 1 blank page for each 15 filled pages.

3.
```
create table sales_south
(stor_id         char(4)      not null,
ord_num          varchar(20)  not null,
date             datetime     not null,
unique clustered (stor_id asc, ord_num desc))
```

   Creates a constraint supported by a unique clustered index; the index order is ascending for *stor_id* and descending for *ord_num*.

4.
```
create table t1
(a  int,
b   char(10))
at "SERVER_A.db1.joe.t1"
```

   Creates a table named *t1* at the remote server SERVER_A and creates a proxy table named *t1* that is mapped to the remote table.

Comments

Specifying Ascending or Descending Ordering in Indexes

- Use the **asc** and **desc** keywords after index column names to specify the sort order for the index. Creating indexes so that columns are in the same order specified in the **order by** clause of queries eliminates the sorting step during query processing. See "Improved Performance for order by Queries" on page 10-21 for more information.

Specifying a Locking Scheme

- To specify the locking scheme for a table, use the keyword **lock** and one of the following locking schemes:
  - **allpages** locking, which locks data pages and the indexes affected by queries
  - **datapages** locking, which locks only data pages
  - **datarows** locking, which locks only data rows

  If you do not specify a locking scheme, the default locking scheme for the server is used. The server-wide default is set with the configuration parameter **lock scheme**.
- The locking scheme for a table can be changed with the **alter table** command. See "Changing a Locking Scheme with alter table" on page 3-3 for more information.

Space Management Properties

- The space management properties **fillfactor**, **max_rows_per_page**, **exp_row_size**, and **reservepagegap** help manage space usage for tables in the following ways:
  - **fillfactor** leaves extra space on pages when indexes are created, but the **fillfactor** is not maintained over time. See "Setting fillfactor Values with sp_chgattribute" on page 4-16 for information on storing **fillfactor** values.
  - **max_rows_per_page** limits the number of rows on a data or index page. Its main use is to improve concurrency in allpages-locked tables, since reducing the number of rows can reduce lock contention. If you specify a **max_rows_per_page** value and **datapages** or **datarows** locking, a warning message is printed. The table is created, and the value is stored in *sysindexes*, but it is applied only if the locking scheme is changed later to **allpages**.

  - **exp_row_size** specifies the expected size of a data row. It applies only to data rows, not to indexes, and applies only to data-only-locked tables that have variable-length columns. It is used to reduce the number of forwarded rows in data-only-locked tables. It is needed mainly for tables where rows have null or short columns when first inserted, bus increase in size as a result of subsequent updates. **exp_row_size** reserves space on the data page for the row to grow to the specified size. If you specify **exp_row_size** when you create an allpages-locked table, a warning message is printed. The table is created, and the value is stored in *sysindexes*, but it is only applied if the locking scheme is changed later to **datapages** or **datarows**.
  - **reservepagegap** specifies the ratio of empty pages to full pages to apply for commands that perform extent allocation. It applies to both data and index pages, in all locking schemes.

- Table 12-6 shows the valid combinations of space management properties and locking scheme. If a **create table** command includes incompatible combinations, a warning message is printed and the table is created. The values are stored in system tables, but are not applied. If the locking scheme for a table changes so that the properties become valid, then they are used.

Table 12-6: Space management properties and locking schemes

Property	allpages	datapages	datarows
max_rows_per_page	Yes	No	No
exp_row_size	No	Yes	Yes
reservepagegap	Yes	Yes	Yes
fillfactor	Yes	Yes	Yes

- Table 12-7 shows the default values and the effects of using default values for the space management properties.

Table 12-7: Defaults and effects of space management properties

Property	Default	Effect of Using the Default
max_rows_per_page	0	Fits as many rows as possible on the page, up to a maximum of 255
exp_row_size	0	Uses the server-wide default value, set with the configuration parameter **default exp_row_size percent**
reservepagegap	0	Leaves no empty pages during extent allocations
fillfactor	0	Fully packs leaf pages, with space left on index pages

Using *exp_row_size*

- If an application inserts short rows into a data-only-locked table and updates them later so that their length increases, use **exp_row_size** to reduce the number of times that rows in data-only-locked tables are forwarded to new locations. See "Reducing Row Forwarding with Expected Row Size" on page 4-1 for more information.

Using *reservepagegap*

- Commands that use large amounts of space allocate new space by allocating an extent rather than allocating single pages. The **reservepagegap** keyword causes these commands to leave empty pages so that subsequent page allocations will take place close to

the page being split or close to the page from which a row is being forwarded. Table 12-8 shows when **reservepagegap** is applied.

Table 12-8: When reservepagegap is applied

Command	Applies to Data Pages	Applies to Index Pages
Fast **bcp**	Yes	Fast **bcp** is not used if indexes exist.
Slow **bcp**	Only for heap tables, not for tables with a clustered index	Extent allocation not performed
select into	Yes	No indexes exist on the target table
create index or **alter table...constraint**	Yes, for clustered indexes	Yes
reorg rebuild	Yes	Yes
alter table...lock (For allpages-locking to data-only-locking, or vice versa)	Yes	Yes

- The **reservepagegap** value for a table is stored in *sysindexes* and is applied when any of the above operations on a table are executed. To change the stored value, use the system procedure **sp_chgattribute**.
- **reservepagegap** is not applied to worktables or sorts on worktables.

Using *at*

- The location information provided by the **at** keyword is the same information that is provided by the **sp_addobjectdef** system procedure. The information is stored in the *sysattributes* table.

Standards and Compliance

Standard	Compliance Level
SQL92	All additions are Transact-SQL extensions

Permissions

create table permission defaults to the Database Owner, who can transfer it to other users. Any user can create temporary tables.

See Also

Commands	**alter table, create existing table, create index, create proxy_table**
System procedures	**sp_chgattribute**

delete

New Functionality

The **readpast** option allows the **delete** command to skip locked rows without blocking.

Syntax

```
delete [[database.]owner.]{table_name | view_name}
[from [[database.]owner.]{view_name [readpast]|
    table_name [readpast]
        [(index {index_name | table_name }
        [ prefetch size ][lru|mru])]}
 [, [[database.]owner.]{view_name [readpast]|
    table_name [readpast]
        [(index {index_name | table_name }
        [ prefetch size ][lru|mru])]} ...]
[where search_conditions] ]
```

New Keywords and Options

readpast – specifies that the **delete** command skip all pages or rows on which incompatible locks are held, without waiting for locks or timing out. For datapages-locked tables, the command skips all rows on pages on which incompatible locks are held; for datarows-locked tables, it skips all rows on which incompatible locks are held.

Examples

1. **delete from authors from authors readpast where state = "CA"**

   Deletes rows from *authors*, skipping any locked rows.

2. **delete stores from stores readpast, authors where stores.city = authors.city**

   Deletes rows from *stores*, skipping any locked rows. If any rows in *authors* are locked, the query blocks on these rows, waiting for the locks to be released.

Comments

Using *readpast*

- The **readpast** option allows **delete** commands on data-only-locked tables to proceed without being blocked by incompatible locks held by other tasks.
  - On datarows-locked tables, readpast skips all rows on which shared, update, or exclusive locks are held by another task.
  - On datapages-locked tables, readpast skips all pages on which shared, update, or exclusive locks are held by another task.
- Commands specifying **readpast** block if there is an exclusive table lock.
- If the **readpast** option is specified for an allpages-locked table, the readpast option is ignored. The command blocks as soon as it finds an incompatible lock.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored. The command executes at level 3. The command blocks on any rows or pages with incompatible locks.
- If the transaction isolation level for a session is 0, a **delete** command using **readpast** does not issue warning messages. For datapages-locked tables, **delete** with **readpast** modifies all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, it affects all rows that are not locked with incompatible locks.
- If the **delete** command applies to a row with two or more text columns, and any text column has an incompatible lock on it, **readpast** locking skips the row.

Standards and Compliance

Standard	Compliance Level
SQL92	**readpast** is a Transact-SQL extension

Permissions

delete permission defaults to the table or view owner, who can transfer it to other users. If **set ansi_permissions** is **on**, you must have **select** permission on all columns appearing in the **where** clause, in addition to the regular permissions required for **delete** statements. By default, **ansi_permissions** is **off**.

delete statistics

Function

Removes statistics from the *sysstatistics* system table.

Syntax

```
delete [shared] statistics table_name [(column_name
   [, column_name]...)]
```

Parameters

shared – removes simulated statistics information from *sysstatistics* in the *master* database.

table_name – removes statistics for all columns in the table.

column_name – removes statistics for the specified column.

Examples

1. `delete statistics titles`

   Delete the densities, selectivities, and histograms for all columns in the *titles* table.

2. `delete statistics titles(pub_id)`

   Deletes densities, selectivities, and histograms for the *pub_id* column in the *titles* table.

3. `delete statistics titles(pub_id, pubdate)`

   Deletes densities, selectivities, and histograms for *pub_id*, *pubdate*, without affecting statistics on the single-column *pub_id* or the single-column *pubdate*.

Comments

- **delete statistics** removes statistics for the specified columns or table from the *sysstatistics* table. It does not affect statistics in the *systabstats* table.
- When you issue the **drop table** command, the corresponding rows in *sysstatistics* are dropped. When you use the **drop index** command, the rows in *sysstatistics* are not deleted. This allows the query optimizer to continue to use index statistics without incurring the overhead of maintaining the index on the table.

◆ ***WARNING!***

Densities, selectivities and histograms are essential to good query optimization. The delete statistics command is provided as a tool to remove statistics not used by the optimizer. If you inadvertently delete statistics needed for query optimization, run update statistics on the table, index or column.

- Loading simulated statistics with the **optdiag** utility command adds a small number of rows to *master..sysstatistics* table. If the simulated statistics are no longer in use, the information in *master..sysstatistics* can be dropped with the **delete shared statistics** command.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only the table owner or a System Administrator can use **delete statistics**.

See Also

Commands	create index, update statistics
Utility Program	optdiag

dump transaction

New Functionality

Dumps only completed transactions.

Syntax

```
dump tran[saction] database_name
to stripe_device [ at backup_server_name ]
    [density = density_value,
     blocksize = number_bytes,
     capacity = number_kilobytes,
     dumpvolume = volume_name,
     file = file_name]
[stripe on stripe_device [ at backup_server_name ]
    [density = density_value,
     blocksize = number_bytes,
     capacity = number_kilobytes,
     dumpvolume = volume_name,
     file = file_name]]
[[stripe on stripe_device [ at backup_server_name ]
    [density = density_value,
     blocksize = number_bytes,
     capacity = number_kilobytes,
     dumpvolume = volume_name,
     file = file_name] ]...]
[with {
    density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    retaindays = number_days,
    [noinit | init],
    notify = {client | operator_console},
    standby_access }]
```

New Parameters

with standby_access – specifies that only completed transactions are to be dumped. The dump continues to the furthest point it can find at which a transaction has just completed and there are no other active transactions. For more detailed information, see “New

with standby_access Option for the dump transaction Command" on page 18-6.

Examples

1. `dump tran inventory_db to dev1 with standby_access`

   Dumps completed transactions from the *inventory_db* transaction log file to device *dev1*.

Comments

- Use the **with standby_access** option to dump transaction logs for loading into a server that acts as a warm standby server for the database.
- When you use **with standby_access** to dump the transaction log, the dump proceeds to the furthest point in the log at which all earlier transactions have completed and there are no records belonging to open transactions.
- You must use **dump tran[saction]...with standby_access** in all situations where you will be loading two or more transaction logs in sequence and you want the database to be online between loads.
- After loading a dump made with the **with standby_access** option, use the **online database** command with the **for standby_access** option to make the database accessible.

◆ ***WARNING!***

If a transaction log contains open transactions and you dump it without the *with standby_access* option, version 11.9.2 does not allow you to load the log, bring the database online, and then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after a load that was originally dumped with standby_access or after loading the entire series.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only System Administrators, users who have been granted the Operator role, and the Database Owner can execute **dump transaction**.

See Also

Commands	online database

lock table

Function

Explicitly locks a table within a transaction.

Syntax

```
lock table table_name in {share | exclusive } mode
      [ wait [ numsecs ] | nowait ]
```

Keywords and Options

table_name – specifies the name of the table to be locked.

share | exclusive – specifies the type of lock, shared or exclusive, to be applied to the table.

wait *numsecs* – specifies the number of seconds to wait, if a lock cannot be acquired immediately. If *numsecs* is omitted, specifies that the **lock table** command should wait until lock is granted.

nowait – causes the command to fail if the lock cannot be acquired immediately.

Examples

1.
```
begin transaction
lock table titles in share mode
```

   Tries to acquire a shared table lock on the *titles* table. If a session-level wait has been set with the **set lock wait** command, the lock table command waits for that period of time; otherwise, the server-level wait period is used.

2.
```
begin transaction
lock table authors in exclusive mode wait 5
```

   Tries to acquire an exclusive table lock on the *authors* table. If the lock cannot be acquired within 5 seconds, the command returns an informational message. Subsequent commands within the transaction continue as they would have without the **lock table** command.

```
3. create procedure bigbatch
   as
   begin transaction
   lock table titles in share mode wait 5
   if @@error = 12207
   begin
         /*
         ** Allow SA to run without the table lock
         ** Other users get an error message
         */
          if (proc_role("sa_role") = 0)
          begin
          print "You cannot run this procedure at
             this time, please try again later"
          rollback transaction
          return 100
          end
   else
          begin
          print "Couldn't obtain table lock,
             proceeding with default locking."
          end
   end
   /* more SQL here */
   commit transaction
```

If a table lock is not acquired within 5 seconds, the procedure checks the user's role. If the procedure is executed by a user with **sa_role**, the procedure prints an advisory message and proceeds without a table lock. If the user does not have **sa_role**, the transaction is rolled back.

Comments

- You can use **lock table** only within a transaction. The table lock is held for the duration of the transaction.
- The behavior of **lock table** depends on the wait-time options that are specified in the command or that are active at the session level or server level.
- If the **wait** and **nowait** option are not specified, the **lock table** command uses either the session-level wait period or the server-level wait period. If a session-level wait has been set using the **set lock wait** command, it is used, otherwise, the server-level wait period is used.
- If the table lock cannot be obtained with the time limit (if any), the **lock table** command returns message 12207. The transaction is not

rolled back. Subsequent commands in the transaction proceed as they would have without the **lock table** command.

- You cannot use **lock table** on system tables or temporary tables.
- You can issue multiple **lock table** commands in the same transaction.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must have **select** access permission on the table to use **lock table in share mode**. You must have either **delete**, **insert**, or **update** access permission on the table to use **lock table in exclusive mode**.

See Also

Commands	set

online database

New Functionality

Brings a database online after loading a transaction log dumped with the **standby_access** option.

Syntax

```
online database database_name [for standby_access]
```

Parameters

database_name – specifies the name of the database to be brought online.

for standby_access – brings the database online on the assumption that the database contains no open transactions.

Examples

1. `online database inventory_db for standby_access`

Brings the database *inventory_db* online. Used after loading *inventory_db* with a transaction-log dump obtained through **dump tran...with standby_access**.

Comments

- **online database...for standby_access** can only be used with a transaction log that was dumped using **dump transaction...with standby_access**. If you use **online database...for standby_access** after loading a transaction log that was dumped without using **dump transaction...with standby access**, the **online database** command will generate an error message and fail.
- You can use **sp_helpdb** to find out whether a database is currently online for standby access.
- See "Changes to the dump, load, and online Commands" on page 18-6 for more information.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Administrator, a Database Owner, or a user with the Operator role can execute **online database for standby_access**.

See Also

Commands	dump transaction

readtext

New Functionality

Allows the **readtext** command to skip rows that have exclusive locks on them, without waiting and without generating a message.

Syntax

```
readtext [[database.]owner.]table_name.column_name
text_pointer offset size
[holdlock | noholdlock]  [shared] [readpast]
[using {bytes | chars | characters}]
[at isolation {read uncommitted | read committed |
    serializable}]
```

New Parameters and Keywords

readpast – specifies that the command should silently skip rows with exclusive locks, without waiting and without generating a message.

Example

```
1. declare @val varbinary(16)
   select @val = textptr(copy) from blurbs readpast
   where au_id = "648-92-1872"
   readtext blurbs.copy @val 1 5 readpast using chars
```

Comments

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.
- The **readpast** option is incompatible with the **holdlock** option. If both are specified in a command, an error is generated and the command terminates.
- If the **readtext** command specifies **at isolation read uncommitted**, the **readpast** option generates a warning, but does not terminate the command.
- If the statement isolation level is set to 3, the **readpast** option generates an error and terminates the command.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored.
- If the session-wide isolation level is 0, the **readpast** option generates a warning, but does not terminate the command.

- For more information on using **readpast**, see "Readpast Locking for Queue Processing" on page 7-5.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

readtext requires **select** permission on the table. **readtext** permission is transferred when **select** permission is transferred.

reorg

Function

Reclaims unused space on pages, removes row forwarding, or rewrites all rows in the table to new pages, depending on the option used.

Syntax

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]
reorg forwarded_rows tablename
   [with {resume,time = no_of_minutes}]
reorg compact tablename
   [with {resume, time = no_of_minutes}]
reorg rebuild tablename
```

Parameters and Keywords

reclaim_space – reclaims unused space left by deletes and updates. For each data page in a table, if there is unused space resulting from committed deletes or row-shortening updates, **reorg reclaim_space** rewrites the current rows contiguously, leaving all unused space at the end of the page. If there are no rows on the page, the page is deallocated.

tablename – specifies the name of the table to be reorganized. If *indexname* is specified, only the index is reorganized.

indexname – specifies the name of the index to be reorganized.

with resume – initiates reorganization from the point at which a previous **reorg** command terminated. Used when the previous **reorg** command specified a time limit (**time =** *no_of_minutes*).

with time = *no_of_minutes* – specifies the number of minutes that the **reorg** command is to run.

forwarded_rows – removes row forwarding.

compact – combines the functions of **reorg reclaim_space** and **reorg forwarded_rows** to both reclaim space and undo row forwarding in the same pass.

rebuild – rewrites all rows in a table to new pages, so that the table is arranged according to its clustered index (if one exists), with all pages conforming to current space management settings and with no forwarded rows and no gaps between rows on a page.

Examples

1. `reorg reclaim_space titles`

   Reclaims unused page space in the **titles** table.

2. `reorg reclaim_space titles titleind`

   Reclaims unused page space in the index **titleind**.

3. `reorg compact titles with time = 120`

   Initiates **reorg compact** on the *titles* table. **reorg** starts at the beginning of the table and continues for 120 minutes. If the **reorg** completes within the time limit, it returns to the beginning of the table and continues until the full time period has elapsed.

4. `reorg compact titles with resume, time = 30`

   Initiates **reorg compact** at the point where the previous **reorg compact** stopped and continues for 30 minutes.

Comments

- The table specified in the **reorg** command must have a datarows or datapages locking scheme.
- You cannot issue the **reorg** command within a transaction.
- **reorg rebuild** requires that you set the database option **select into/bulkcopy/pllsort** to **true** and run **checkpoint** in the database.
- **reorg rebuild** requires additional disk space equal to the size of the table and its indexes. You can find out how much space a table currently occupies by using **sp_spaceused**. You can use **sp_helpsegment** to check the amount of space available.
- After running **reorg rebuild**, you must dump the database before you can dump the transaction log.
- For information on setting space management properties that are applied during reorg rebuild commands, see Chapter 4, “Setting Space Management Properties.”
- See also Chapter 17, “Using the reorg Command.”

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

You must be a System Administrator or the object owner to issue the **reorg** command.

See Also

System procedures	sp_chgattribute

select

New Functionality

Allows specifying the locking scheme with select into; allows a **select** command to skip rows that have exclusive locks on them without waiting and without generating a message.

Syntax

```
select [all | distinct] select_list
[into [[database.]owner.]table_name
    [lock {datarows | datapages | allpages }]
        [with { max_rows_per_page = num_rows,
            exp_row_size = num_bytes,
            reservepagegap = num_pages } ] ]
[from [[database.]owner.]{view_name|table_name
[(index {index_name | table_name }
        [parallel [degree_of_parallelism]]
        [prefetch size ][lru|mru])]}
    [holdlock | noholdlock] [readpast] [shared]
 [,[[database.]owner.]{view_name|table_name
    [(index {index_name | table_name }
        [parallel [degree_of_parallelism]]
        [prefetch size ][lru|mru])]}
   [holdlock | noholdlock] [readpast] [shared]]...]
[where search_conditions]
[group by [all] aggregate_free_expression
    [, aggregate_free_expression]... ]
[having search_conditions]
[order by
  {[[[database.]owner.]{table_name.|view_name.}]
    column_name | select_list_number | expression}
        [asc | desc]
  [,{[[[database.]owner.]{table_name|view_name.}]
    column_name | select_list_number | expression}
        [asc | desc]]...]
[compute row_aggregate(column_name)
        [, row_aggregate(column_name)]...
    [by column_name [, column_name]...]]
[for {read only | update [of column_name_list]}]
[at isolation {
    [ read uncommitted | 0 ] |
    [ read committed | 1 ] |
    [ repeatable read | 2 ]|
    [ serializable | 3 ] } ]
|[for browse]
```

New Parameters and Keywords

lock datarows | datapages | allpages – specifies the locking scheme to be used for a table created with a **select into** command. The default is the server-wide setting for the configuration parameter **lock scheme**.

max_rows_per_page – limits the number of rows on data pages for a table created with **select into**. Unlike **fillfactor**, the **max_rows_per_page** value is maintained when data is inserted or deleted. **max_rows_per_page** is not supported on data-only-locked tables.

exp_row_size = *num_bytes* – specifies the expected row size for a table created with the **select into** command. Valid only for datarows and datapages locking schemes and only for tables that have variable-length rows. Valid values are 0, 1, and any value greater than the minimum row length and less than the maximum row length for the table. The default value is 0, which means that a server-wide default is used.

reservepagegap = *num_pages* – specifies a ratio of filled pages to empty pages that is to be left as **select into** allocates extents to store data. This option is valid only for the **select into** command. For each specified *num_pages*, one empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

readpast – specifies that the query should silently skip rows with exclusive locks, without waiting and without generating a message.

repeatable read | 2 – specifies transaction isolation level 2 (repeatable reads) for the query.

Examples

1. ```
   select title_id, title, price
   into bus_titles
   lock datarows with reservepagegap = 10
   from titles
   where type = "business"
   ```

   Specifies the locking scheme and the reserve page gap for **select into**.

2. ```
   select title, price
   from titles readpast
       where type = "news"
       and price between $20 and $30
   ```

Selects only the rows that are not exclusively locked. If any other user has an exclusive lock on a qualifying row, that row is not returned.

```
3. begin tran
   select type, avg(price)
       from titles
       group by type
   at isolation repeatable read
```

Selects from *titles* using the **repeatable read** isolation level. No other user can change values in or delete the affected rows until the transaction completes.

Comments

Specifying a Lock Scheme with *select...into*

- The **lock** option, used with **select...into**, allows you to specify the locking scheme for the table created by the command. If you do not specify a locking scheme, the default locking scheme, as set by the configuration parameter **lock scheme**, is applied. See Chapter 3, "Creating and Managing Data-Only-Locked Tables," for more information on locking schemes.
- When you use the **lock** option, you can also specify the space management properties **max_rows_per_page**, **exp_row_size**, and **reservepagegap**. See "Space Management Properties" on page 12-25 for information.
- You can change the space management properties for a table created with **select into**, using the **sp_chgattribute** system procedure.

The *readpast* Option

- The **readpast** option allows a **select** command to access the specified table without being blocked by incompatible locks held by other tasks. **readpast** queries can only be performed on data-only-locked tables.
- If the **readpast** option is specified for an allpages-locked table, the **readpast** option is ignored. The command operates at the isolation level specified for the command or session. If the isolation level is 0, dirty reads are performed, and the command returns values from locked rows and does not block. If the isolation level is 1 or 3, the command blocks when pages with incompatible locks must be read.

- The interactions of session-level isolation levels and **readpast** on a table in a **select** command are shown in Table 12-9.

Table 12-9: Effects of session-level isolation levels and readpast

Session Isolation Level	Effects
0, **read uncommitted** (dirty reads)	**readpast** is ignored, and rows containing uncommitted transactions are returned to the user. A warning message is printed.
1, **read committed**	Rows or pages with incompatible locks are skipped; no locks are held on the rows or pages read
2, **repeatable read**	Rows or pages with incompatible locks skipped; shared locks are held on all rows or pages that are read until the end of the statement or transaction.
3, **serializable**	**readpast** is ignored, and the command executes at level 3. The command blocks on any rows or pages with incompatible locks.

- **select** commands that specify **readpast** fail with an error message if they also include any of the following:
  - An **at isolation** clause, specifying **0** or **read uncommitted**
  - An **at isolation** clause, specifying **3** or **serializable**
  - The **holdlock** keyword on the same table
- If **at isolation 2** or **at isolation repeatable read** is specified in a **select** query that specifies **readpast**, shared locks are held on the **readpast** tables until the statement or transaction completes.
- If a **select** command with the **readpast** option encounters a text column that has an incompatible lock on it, **readpast** locking retrieves the row, but returns the text column with a value of **null**. No distinction is made, in this case, between a text column containing a null value and a null value returned because the column is locked.

Repeatable Reads

- The repeatable reads isolation level, also known as transaction isolation level 2, holds locks on all pages read by the statement until the transaction completes.

Standards and Compliance

Standard	Compliance Level
SQL92	All 11.9.2 changes are Transact-SQL extensions

Permissions

select permission defaults to the owner of the table or view, who can transfer it to other users.

See Also

Commands	create index
System Procedures	sp_chgattribute

set

New Functionality

Specifies the length of time that a query waits to acquire a lock before aborting and returning an error message.

Sets the transaction isolation level to isolation level 2, repeatable reads.

Specifies that the query should be optimized using simulated statistics.

Syntax

```
set lock { wait [ numsecs ] | nowait }
set transaction isolation level {
    [ read uncommitted | 0 ] |
    [ read committed | 1 ] |
    [ repeatable read | 2 ]|
    [ serializable | 3 ] }
set statistics simulate { on | off }
```

New Parameters and Keywords

lock wait – specifies the length of time that a command waits to acquire locks before aborting and returning an error.

numsecs – specifies the number of seconds a command is to wait to acquire a lock. Valid values are from 0 to 2147483647, the maximum value for an integer.

lock nowait – specifies that if a command cannot acquire a lock immediately, it returns an error and fails. **set lock nowait** is equivalent to **set lock wait 0**.

repeatable read | 2 – prevents nonrepeatable reads.

statistics simulate – specifies that the optimizer should use simulated statistics to optimize the query.

Examples

1. set lock wait 5

Subsequent commands in the session or stored procedure wait 5 seconds to acquire locks before generating an error message and failing.

2. set lock nowait

Subsequent commands in the session or stored procedure return an error and fail if they cannot get requested locks immediately.

3. `set lock wait`

   Subsequent commands in the current session or stored procedure will wait indefinitely long to acquire locks.

4. `set transaction isolation level 2`

   All subsequent queries in the session will run at the repeatable reads transaction isolation level.

Comments

lock wait

- By default, an Adaptive Server task that cannot immediately acquire a lock waits until incompatible locks are released and then continues processing. This is equivalent to **set lock wait** with no value specified in the *numsecs* parameter.
- You can set a server-wide lock wait period by using the **sp_configure** system procedure with the **lock wait period** option. See “lock wait period” on page 16-5.
- A lock wait period defined at the session level or in a stored procedure with the **set lock** command overrides a server-level lock-wait period.
- If **set lock wait** is used by itself, with no value for *numsecs*, all subsequent commands in the current session wait indefinitely to acquire requested locks.
- The **sp_sysmon** procedure reports the number of times that tasks waiting for a lock could not acquire the lock within the waiting period. See “Lock Timeout Information” on page 20-3.

Repeatable-Reads Transaction Isolation Level

- The repeatable-reads isolation level, also known as transaction isolation level 2, holds locks on all pages read by the statement until the transaction completes. See “Repeatable Reads Isolation Level” on page 7-9 for more information.
- A nonrepeatable read occurs when one transaction reads rows from a table and a second transaction is able to modify the same rows and commit the changes before the first transaction completes. If the first transaction rereads the rows, they will have different values, so the initial read is not repeatable. Repeatable

reads hold shared locks for the duration of a transaction, blocking transactions that update the locked rows or rows on the locked pages.

Using Simulated Statistics

- You can load simulated statistics into a database using the **simulate** mode of the **optdiag** utility program. If **set statistics simulate on** has been issued in a session, queries are optimized using simulated statistics, rather than the actual statistics for a table.
- For more information on simulated statistics, see "Using Simulated Statistics" on page 8-29.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

set lock wait, **set statistics simulate**, and **set transaction isolation level** permission defaults to all users, and no special permissions are required to use it.

See Also

Commands	lock table
Utility Program	optdiag

update

New Functionality

The **readpast** clause allows the **update** command to skip locked rows without blocking.

Syntax

```
update [[database.]owner.]{table_name | view_name}
set [[[database.]owner.]{table_name.|view_name.}]
    column_name1 =
        {expression1|NULL|(select_statement)} |
    variable_name1 =
        {expression1|NULL|(select_statement)}
    [, column_name2 =
        {expression2|NULL|(select_statement)}]... |
    [, variable_name2 =
        {expression2|NULL|(select_statement)}]...

[from [[database.]owner.]{view_name [readpast]|
    table_name [readpast]
        [(index {index_name | table_name }
        [ prefetch size ][lru|mru])]}
     [,[[database.]owner.]{view_name [readpast]|
    table_name [readpast]
        [(index {index_name | table_name }
        [ prefetch size ][lru|mru])]}]
...]
[where search_conditions]
```

New Keywords and Options

readpast – causes the **update** command to modify unlocked rows only on datarows-locked tables, or rows on unlocked pages, for datapages-locked tables. **update...readpast** silently skips locked rows or pages rather than waiting for the locks to be released.

Examples

```
1. update salesdetail set discount = 40
       from salesdetail readpast
   where title_id like "BU1032"
       and qty > 100
```

Updates rows on which another task does not hold a lock.

Comments

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.
- The **readpast** option is incompatible with the **holdlock** option. If both are specified in the same **select** command, an error is generated and the command terminates.
- If the session-wide isolation level is 3, the **readpast** option is ignored.
- If the transaction isolation level for a session is 0, **update** commands using **readpast** do not issue warning messages. For datapages-locked tables, these commands modify all rows on all pages that are not locked with incompatible locks. For datarows-locked tables, they affect all rows that are not locked with incompatible locks.
- If an **update** command with the **readpast** option applies to two or more text columns, and the first text column checked has an incompatible lock on it, readpast locking skips the row. If the column does not have an incompatible lock, the command acquires a lock and modifies the column. Then, if any subsequent text column in the row has an incompatible lock on it, the command blocks until it can obtain a lock and modify the column.
- For more information on **readpast** locking, see "Readpast Locking for Queue Processing" on page 7-5.

Standards and Compliance

Standard	Compliance Level
SQL92	**readpast** is Transact-SQL extension

Permissions

If **set ansi_permissions** is **on**, you need **update** permission on the table being updated and, in addition, you must have **select** permission on all columns appearing in the **where** clause and on all columns following the **set** clause. By default, **ansi_permissions** is **off**.

No special permission is required for using the **readpast** option.

update statistics

New Functionality

Updates information about the distribution of key values in specified indexes or for specified columns, for all columns in an index or for all columns in a table; allows specifying the number of steps for a histogram.

Syntax

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

update all statistics table_name

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

New Keywords and Options

column_list – is a comma-separated list of columns.

using *step* **values** – specifies the number of histogram steps. The default value is 20, for columns where no statistics exist. If statistics for a column already exist in *sysstatistics*, the default value is the current number of steps.

with consumers = *consumers* – specifies the number of consumer processes to be used for a sort when *column_list* is provided and parallel query processing is enabled.

index – specifies that statistics for all columns in an index are to be updated.

all – specifies that statistics for all columns of a table and partition statistics are to be updated.

Examples

1. **update statistics titles (price) using 40 values**

   Generates statistics for the *price* column of the *titles* table.

2. **update index statistics authors**

Generates statistics for all columns in all indexes of the *authors* table.

3. `update index statistics authors au_names_ix`

   Generates statistics for all columns in the *au_names_ix* index of the *authors* table.

4. `update all statistics titles`

   Generates statistics for all columns in the *titles* table. If *titles* is partitioned, this command also updates partition statistics.

Comments

- The **update statistics** command, when used with a table name and an index name, updates statistics for the leading column of an index. If **update statistics** is used with just a table name, it updates statistics for the leading columns of all indexes on the table.
- The **update index statistics** command, when used with a table name and an index name, updates statistics for all columns in the specified index. If **update index statistics** is used with just a table name, it updates statistics for all columns in all indexes of the table.
- The **update all statistics** command updates statistics for all columns in a table and updates partition statistics, if the table is partitioned.
- Specifying the name of an unindexed column or the nonleading column of an index generates statistics for that column without creating an index.
- Specifying more than one column in a column list generates or updates a histogram for the first column, and density statistics for all prefix subsets of the list of columns.
- If you use **update statistics** to generate statistics for a column or list of columns, **update statistics** must scan the table and perform a sort. See “Scan Types, Sort Requirements, and Locking During update statistics” on page 9-7 for more information.
- The **with consumers** clause 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. For more information, see Chapter 15, “Parallel Sorting,” in the *Performance and Tuning Guide*.

- Table 12-10 shows the types of scans performed during **update statistics**, the types of locks acquired, and when sorts are needed.

Table 12-10:Locking, scans, and sorts 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		
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

- 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)
  ```

- 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.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

update statistics permission defaults to the table owner and is not transferable. The command can also be executed by the Database Owner, who can impersonate the table owner by running the **setuser** command.

See Also

Commands	delete statistics

writetext

New Functionality

Skip locked rows without blocking.

Syntax

```
writetext [[database.]owner.]table_name.column_name
text_pointer [readpast] [with log] data
```

New Parameters and Keywords

readpast – specifies that the command should modify only unlocked rows. If the **writetext** command finds locked rows, it skips them, rather than waiting for the locks to be released.

Examples

```
1. declare @val varbinary(16)
   select @val = textptr(copy)
   from blurbs readpast
       where au_id = "409-56-7008"
   writetext blurbs.copy @val readpast with log
   "hello world"
```

Comments

- The **readpast** option applies only to data-only-locked tables. **readpast** is ignored if it is specified for an allpages-locked table.
- If the session-wide isolation level is 3, the **readpast** option is silently ignored.
- If the transaction isolation level for a session is 0, **writetext** commands using **readpast** do not issue warning messages. These commands at session isolation level 0 modify the specified text column if the text column is not locked with incompatible locks.
- For more information on **readpast** locking, see “Readpast Locking for Queue Processing” on page 7-5.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

writetext permission defaults to the table owner, who can transfer it to other users.

13 New and Changed Functions

This chapter describes the following new and changed Transact-SQL functions:

index_colorder

Function

Returns the column order.

Syntax

```
index_colorder (object_name, index_id, key_#
   [, user_id])
```

Arguments

object_name – is the name of a table or view. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.

index_id – is the number of *object_name*'s index. This number is the same as the value of *sysindexes.indid*.

key_# – is a key in the index. Valid values are 1 and the number of keys in the index. The number of keys is stored in *sysindexes.keycnt*.

user_id – is the owner of *object_name*. If you do not specify *user_id*, it defaults to the caller's user ID.

Examples

```
1. select name, index_colorder("sales", indid, 2)
   from sysindexes
   where id = object_id ("sales")
   and indid > 0
```

```
name
------------------------- -------------------------
salesind                  DESC
```

Returns "DESC" because the *salesind* index on the *sales* table is in descending order.

Comments

- **index_colorder**, a system function, returns "ASC" for columns in ascending order or "DESC" for columns in descending order.
- **index_colorder** returns NULL if *object_name* is not a table name or if *key_#* is not a valid key number.

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Any user can execute **index_colorder.**

lct_admin

New Functionality

Returns the current value of the last-chance threshold.

Aborts transactions in a transaction log that has reached its last-chance threshold.

Syntax

```
lct_admin({{"lastchance" | "logfull" }, database_id
|"reserve", {log_pages | 0 }
| "abort", process-id [, database-id]})
```

Arguments

reserve – obtains either the current value of the last-chance threshold or the number of log pages required for dumping a transaction log of a specified size.

0 – returns the current value of the last-chance threshold. The size of the last-chance threshold in a database with separate log and data segments does not vary dynamically. It has a fixed value, based on the size of the transaction log. The last-chance threshold varies dynamically in a database with mixed log and data segments.

abort – aborts transactions in a database where the transaction log has reached its last-chance threshold. Only transactions in LOG SUSPEND mode can be aborted.

process-id – The ID (*spid*) of a process in log-suspend mode. A process is placed in log-suspend mode when it has open transactions in a transaction log that has reached its last-chance threshold (LCT).

database-id – the ID of a database whose transaction log has reached its LCT. If *process-id* is **0**, all open transactions in the specified database are terminated.

Examples

1. **`select lct_admin("reserve",0)`**

   Returns the current last-chance threshold of the transaction log in the database from which the command was issued.

2. **`select lct_admin("abort", 83)`**

Aborts transactions belonging to process 83. The process must be in log-suspend mode. Only transactions in a transaction log that has reached its LCT are terminated.

3. `select lct_admin("abort", 0, 5)`

   Aborts all open transactions in the database with database ID 5.

Comments

- To terminate the oldest open transaction in a transaction log that has reached its LCT, enter the ID of the process that initiated the transaction. See "Using lct_admin abort" on page 18-9.
- To terminate all open transactions in a transaction log that has reached its LCT, enter **0** as the *process_id*, and specify a database ID in the *database-id* parameter.
- For more information on using **lct_admin abort**, see "Using lct_admin abort" on page 18-9
- For more information on using **lct_admin reserve**, see "Using lct_admin reserve to Get the Current Last-Chance Threshold" on page 18-11.
- The **lct_admin unsuspend** command is not supported in Adaptive Server 11.9.2

Standards and Compliance

Standard	Compliance Level
SQL92	Transact-SQL extension

Permissions

Only a System Administrator can execute **lct_admin abort**.

14 New and Changed System Procedures

This chapter describes the following system procedures:

sp_chgattribute

New Functionality

Changes the **max_rows_per_page**, **fillfactor**, **reservepagegap**, or **exp_row_size** value for future space allocations of a table or an index.

Sets the **concurrency_opt_threshold** for a table.

Syntax

```
sp_chgattribute objname, {"max_rows_per_page" |
"fillfactor" | "reservepagegap" | "exp_row_size"
concurrency_opt_threshold }, optvalue
```

Parameters

objname – is the name of the table or index.

fillfactor – specifies how full Adaptive Server will make each page when it is re-creating an index or copying table pages as a result of a **reorg rebuild** command or an **alter table** command to change the locking scheme. The **fillfactor** percentage is relevant only at the time the index is rebuilt. Valid values are 0–100.

reservepagegap – specifies the ratio of filled pages to empty pages that are to be left during extent I/O allocation operations. For each specified *num_pages*, an empty page is left for future expansion of the table. Valid values are 0–255. The default value is 0.

exp_row_size – reserves a specified amount of space for the rows in data-only locked tables. Use this option to reduce the number of rows being forwarded, which can be expensive during updates. Valid values are 0, 1, and any value between the minimum and maximum row length for the table. 0 means a server-wide setting is applied, and 1 means to fully pack the rows on the data pages.

concurrency_opt_threshold – specifies the table size, in pages, at which access to a data-only-locked table should begin optimizing for reducing I/O, rather than for concurrency. If the table is smaller than the number of pages specified by **concurrency_opt_threshold**, the query is optimized for concurrency by always using available indexes; if the table is larger than the number of pages specified by **concurrency_opt_threshold**, the query is optimized for I/O instead. Valid values are 0–32767. Setting the value to 0 disables concurrency optimization. The default is 15 pages.

optvalue – is the new value. Valid values and default values depend on which parameter is specified.

Examples

1. `sp_chgattribute authors, "exp_row_size", 120`

   Sets the **exp_row_size** to 120 for the *authors* table for all future space allocations.

2. `sp_chgattribute "titles.titleidind", "reservepagegap", 16`

   Sets the **reservepagegap** to 16 for the *titleidind* index for all future space allocations.

3. `sp_chgattribute "titles.title_ix", "fillfactor", 90`

   Specifies a **fillfactor** of 90 percent for pages in *title_ix*.

4. `sp_chgattribute "titles", concurrency_opt_threshold, 0`

   Turns off concurrency optimization for the *titles* table.

Comments

- **sp_chgattribute** changes the **max_rows_per_page**, **fillfactor**, **reservepagegap**, or **exp_row_size** value for future space allocations or data modifications of the table or index. It does not affect the space allocations of existing data pages. You can change these values for an object only in the current database.
- If you specify an incorrect value for **max_rows_per_page**, **fillfactor**, **reservepagegap**, or **exp_row_size**, **sp_chgattribute** returns an error message specifying the valid values.
- For more information on **fillfactor** and **max_rows_per_page**,Chapter 4, “Setting Space Management Properties.”
- For more information on **exp_row_size**, see “Reducing Row Forwarding with Expected Row Size” on page 4-1.
- For more information on **reservepagegap**, see “Leaving Space for Forwarded Rows and Inserts” on page 4-8.
- For more information on **concurrency_opt_threshold**, see “Concurrency Optimization for Small Tables” on page 10-7.

Permissions

Only the object owner can execute **sp_chgattribute**.

Tables Used

sysindexes, sysobjects, systabstats

See Also

Commands	**alter table**, **create index**, **create table**
System procedures	**sp_helpindex**

sp_dbrecovery_order

New Functionality

Specifies the order in which user databases are recovered and lists the user-defined recovery order of a database or all databases.

Syntax

```
sp_dbrecovery_order
   [database_name [, rec_order [, force]]]
```

Parameters

database_name – The name of the database being assigned a recovery order or the database whose user-defined recovery order is to be listed.

rec_order – The order in which the database is to be recovered. A *rec_order* of **-1** deletes a specified database from the user-defined recovery sequence.

force – allows the user to insert a database into an existing recovery sequence without putting it at the end.

Examples

1. sp_dbrecovery_order pubs2, 1

Makes the *pubs2* database the first user database to be recovered following a system failure.

2. sp_dbrecovery_order pubs3, 3, force

Inserts the *pubs3* database into third position in a user-defined recovery sequence. If another database was initially in third position, it is moved to fourth position, and all databases following it are moved accordingly.

3. sp_dbrecovery_order pubs2, -1

Removes the *pubs2* database from the user-defined recovery sequence. Subsequently, *pubs2* will be recovered after all databases with a user-specified recovery order have recovered.

4. sp_dbrecovery_order

Lists the current recovery order of all databases with a recovery order assigned through **sp_dbrecovery_order**.

Comments

- You must be in the *master* database to use **sp_dbrecovery_order** to enter or modify a user-specified recovery order. You can list the user-defined recovery order of databases from any database.
- To change the user-defined recovery position of a database, you must first use **sp_dbrecovery_order** to delete the database from the recovery sequence and then use **sp_dbrecovery_order** to insert it into a new position.
- System databases are always recovered before user databases. The system databases and their recovery order are:
  - *master*
  - *model*
  - *tempdb*
  - *sybsystemdb*
  - *sybsecurity*
  - *sybsystemprocs*
- If no database is assigned a recovery order through **sp_dbrecovery_order**, all user databases are recovered in order, by database ID, after system databases.
- If *database_name* is specified, but no *rec_order* is given, **sp_dbrecovery_order** shows the user-defined recovery position of the specified database.
- If *database_name* is not specified, **sp_dbrecovery_order** lists the recovery order of all databases with a user-assigned recovery order.
- The order of recovery assigned through **sp_dbrecovery_order** must be consecutive, starting with **1** and containing no gaps between values. The first database assigned a recovery order must be assigned a *rec_order* of **1**. If three databases have been assigned a recovery order of **1**, **2**, and **3**, you cannot assign the next database a recovery order of **5**.

Permissions

Only a System Administrator can use **sp_dbrecovery_order** to specify a recovery order.

Tables Used

master..sysattributes

sp_droprowlockpromote

Function

Removes row lock promotion threshold values from a database or table.

Syntax

```
sp_droprowlockpromote {"database" | "table"}, objname
```

Parameters

database | **table** – specifies whether to remove the row lock promotion thresholds from a database or table.

objname – is the name of the database or table from which to remove the row lock promotion thresholds.

Examples

```
1. sp_droprowlockpromote "table", "sales"
```

Removes the row lock promotion values from the *sales* table. Lock promotion for *sales* now uses the database or server-wide values.

Comments

- Use **sp_droprowlockpromote** to drop row lock promotion values set with **sp_setrowlockpromote.**
- When you drop a database's row lock promotion thresholds, datarows-locked tables that do not have row lock promotion thresholds configured use the server-wide values. Use **sp_configure** to check the value of the row lock promotion configuration parameters.
- When a table's row lock promotion values are dropped, Adaptive Server uses the database's row lock promotion thresholds, if they are configured, or the server-wide values, if no thresholds are set for the database.
- To change the lock promotion thresholds for a database, you must be using the *master* database. To change the lock promotion thresholds for a table in a database, you must be using the database where the table resides.
- Server-wide values can be changed with **sp_setrowlockpromote.** This changes the values in the row lock promotion configuration

parameters, so there is no corresponding server option for **sp_droprowlockpromote**.

Permissions

Only a System Administrator can execute **sp_droprowlockpromote**.

Tables Used

master.dbo.sysattributes, sysobjects

See Also

System procedures	**sp_setrowlockpromote**

sp_flushstats

Function

Flushes statistics from in-memory storage to the *systabstats* system table.

Syntax

```
sp_flushstats objname
```

Parameters

objname – is the name of a table.

Examples

```
1. sp_flushstats titles
```

Flushes statistics for the *titles* table.

Comments

- Some statistics in the *systabstats* table are updated in in-memory storage locations and flushed to *systabstats* periodically, to reduce overhead and contention on *systabstats*.
- If you query *systabstats* using SQL, executing **sp_flushstats** guarantees that in-memory statistics are flushed to *systabstats*.
- The **optdiag** command always flushes in-memory statistics before displaying output.
- The statistics in *sysstatistics* are changed only by data definition language commands and do not require the use of **sp_flushstats**.

Permissions

Only a System Administrator can execute **sp_flushstats**.

sp_forceonline_object

Function

Provides access to an index previously marked suspect by recovery.

Syntax

```
sp_forceonline_object dbname, objname, indid,
   {sa_on | sa_off | all_users} [, no_print]
```

Parameters

dbname – is the name of the database containing the index to be brought online.

objname – is the name of the table.

indid - is the index ID of the suspect index being brought online.

sa_on – allows only users with the **sa_role** to access the specified index.

sa_off – revokes access privileges created by a previous invocation of **sp_forceonline_object** with **sa_on**.

all_users – allows all users to access the specified index.

no_print - skips printing a list of other suspect objects after the specified object is brought online.

Examples

1. **`sp_forceonline_object pubs2, titles, 3 , sa_on`**

   Allows a System Administrator to access the index with indid 3 on the *titles* table in the *pubs2* database.

2. **`sp_forceonline_object pubs2, titles 3, sa_off`**

   Revokes access to the index from the System Administrator. Now, no one has access to this index.

3. **`sp_forceonline_object pubs2, titles, 3, all_users`**

   Allows all users to access the index on the *titles* table in the *pubs2* database.

Comments

- If an index on a data-only-locked table has suspect pages, the entire index is taken offline during recovery. Offline indexes are

not considered by the query optimizer. Indexes on allpages-locked tables are not taken completely offline during recovery; only individual pages of these indexes are taken offline. These pages can be brought online with **sp_forceonline_page**.

- Use **sp_listsuspect_object** to see a list of databases that are offline.
- To repair a suspect index, use **sp_forceonline_object** with **sa_on** access. Then, drop and re-create the index.

➤ ***Note***

If the index is on *systabstats* or *sysstatistics* (the only data-only-locked system tables) call Sybase Technical Support for assistance.

- **sp_forceonline_object** with **all_users** cannot be reversed. When an index has been brought online for all users, you cannot take it offline again.
- An index that is forced online is not necessarily repaired. Corrupt indexes can be forced online. Adaptive Server does not perform any consistency checks on indexes that are forced online.
- **sp_forceonline_object** cannot be used in a transaction.
- **sp_forceonline_object** works only for databases in which the recovery fault isolation mode is "page." Use **sp_setsuspect_granularity** to display the recovery fault isolation mode for a database.
- To bring all of a database's offline pages and indexes online in a single command, use **sp_forceonline_db**.
- See "Fault Isolation During Recovery" on page 20-7 of the System Administration Guide for more information on recovery fault isolation.

See Also

System procedures	**sp_listsuspect_object**

sp_help

New Functionality

Reports the locking scheme, expected row size, reserve page gap, and row lock promotion setting for tables.

Syntax

```
sp_help table_name
```

Parameters

table_name – is the name of the table to report on.

Comments

- **sp_help** displays the following new settings:
  - The locking scheme, which can be set with **create table** and changed with **alter table**
  - The expected row size, which can be set with **create table** and changed with **sp_chgattribute**
  - The reserve page gap, which can be set with **create table** and changed with **sp_chgattribute**
  - The row lock promotion settings, which can be set or changed with **sp_setrowlockpromote** and dropped with **sp_droprowlockpromote**
- **sp_help** includes the report from **sp_helpindex**, which shows the order of the keys used to create the index and the space management properties.

Examples

1. sp_help titles

```
Name         Owner                    Type
------------ ------------------------ ----------------------
titles       dbo                      user table

Data_located_on_segment        When_created
------------------------------ --------------------------
default                         Dec  6 1997 12:07PM

Column_name     Type    Length Prec Scale Nulls Default_name Rule_name Identity
--------------- ------- ------ ---- ----- ----- ------------ --------- --------
title_id        tid          6 NULL  NULL     0 NULL         NULL             0
title           varchar     80 NULL  NULL     0 NULL         NULL             0
type            char        12 NULL  NULL     0 typedflt     NULL             0
pub_id          char         4 NULL  NULL     1 NULL         NULL             0
```

```
price           money        8 NULL  NULL     1 NULL         NULL             0
advance         money        8 NULL  NULL     1 NULL         NULL             0
total_sales     int          4 NULL  NULL     1 NULL         NULL             0
notes           varchar    200 NULL  NULL     1 NULL         NULL             0
pubdate         datetime     8 NULL  NULL     0 datedflt     NULL             0
contract        bit          1 NULL  NULL     0 NULL         NULL             0

attribute_class        attribute              int_value
         char_value
         comments
----------------------------- ----------------------------- -----------
lock strategy                   row lock promotion                    NULL
         PCT = 95, LWM = 300, HWM = 300
         NULL

index_name           index_description
        index_keys
        index_max_rows_per_page index_fillfactor index_reservepagegap
------------------- ------------------------------------------------------
       ---------------------- --------------- --------------------
titleidind           clustered, unique located on default
         title_id
                               0               0                    0
titleind             nonclustered located on default
         title
                               0               0                    0
type_price           nonclustered located on default
         type, price DESC
                               0               0                    0
No defined keys for this object.
Msg 18085, Level 16, State 1:
Server 'sagan', Procedure 'sp_helpartition', Line 83:
Object is not partitioned.
Lock scheme Datarows


(2 rows affected)
exp_row_size reservedpagegap
------------ ---------------
         224              16
```

Reports on the *titles* table. Information that is new in this version is shown in bold.

See Also

Commands	**alter table, create table**
System procedures	**sp_chgattribute, sp_droprowlockpromote, sp_setrowlockpromote**

sp_helpdb

New Functionality

Reports row lock promotion thresholds for a database.

Syntax

```
sp_helpdb [dbname]
```

Parameters

dbname – is the name of the database to report on.

Comments

- **sp_helpdb** displays the database-specific row lock promotion attribute, if one is defined for the database:

```
name                          attribute_class
attribute                     int_value
         char_value
         comments
-------------------------------------------------------------
pubtune                       lock strategy
      row lock promotion                    NULL
      PCT = 95, LWM = 300, HWM = 300
```

sp_helpindex

New Functionality

Reports new information about the indexes created on a table.

Syntax

```
sp_helpindex objname
```

Parameters

objname – is the name of a table in the current database.

Comments

- **sp_helpindex** displays:
  - Information about clustered indexes on data-only locked tables

    The index ID (*indid)* of a clustered index in data-only locked tables is not equal to 1.
  - The column order of the keys, to indicate whether they are in ascending or descending order.
  - Space manage property values
  - The key column name followed by the order. Only descending order is displayed. For example, if there is an index on column a ASC, b DESC, c ASC, "index_keys" shows "a, b DESC, c".

Examples

1. sp_helpindex titles

```
index_name       index_description
   index_keys
   index_max_rows_per_page index_fillfactor index_reservepagegap
---------------- ---------------------------------------------
title_id_ix      nonclustered, unique located on default
    title_id
                         0               0                    0
publ_ix          nonclustered located on default
    pub_id, pubdate DESC
                         0               0                    8
title_ix         clustered, allow duplicate rows located on default
    title
                         0              90                    0
```

The index on *publ_ix* was created with *pub_id* in ascending order and *pubdate* in descending order.

sp_listsuspect_object

Function

Lists all indexes in a database that are currently offline because of corruption detected on recovery.

Syntax

```
sp_listsuspect_object [dbname]
```

Parameters

dbname – is the name of the database.

Examples

1. `sp_listsuspect_object`

   Lists the suspect indexes in the current database.

2. `sp_listsuspect_object pubs2`

   Lists the suspect indexes in the *pubs2* database.

Comments

- If an index on a data-only-locked table has suspect pages, the entire index is taken offline during recovery. Offline indexes are not considered by the query optimizer.
- Use the system procedure **sp_forceonline_object** to bring an offline index online for repair.
- Indexes on allpages-locked tables are not taken completely offline during recovery; only individual pages of these indexes are taken offline. These pages can be brought online with **sp_forceonline_page**.
- **sp_listsuspect_object** lists the database name, object ID, object name, index ID, and access status for every suspect index in the specified database or, if *dbname* is omitted, in the current user database.
- A value of SA_ONLY in the *access* column means that the index has been forced online for System Administrator use only. A value of BLOCK_ALL means that the index is offline for everyone.

- See “Fault Isolation During Recovery” on page 20-7 of the System Administration Guide for more information on recovery fault isolation.

Permissions

Any user can execute **sp_listsuspect_object**.

See Also

System procedures	**sp_forceonline_object**

sp_lock

New Functionality

Reports additional information about processes that currently hold locks.

Syntax

```
sp_lock [spid1 [, spid2]]
```

Parameters

spid1 – is the server process ID number from the *master.dbo.sysprocesses* table. Run **sp_who** to get the *spid* of the locking process.

spid2 – is another server process ID number to check for locks.

Comments

Changes to *sp_lock* Output

sp_lock output includes a new column, *row*, that displays the row number for row-level locks. In addition, **sp_lock** output now displays:

- New lock types:
  - "Sh_row" indicates shared row locks
  - "Update_row" indicates update row locks
  - "Ex_row" indicates exclusive row locks
- Additional information in the *context* column:
  - "Ind pg" indicates locks on index pages (allpages-locked tables only)
  - "Inf key" indicates an infinity key lock (for certain range queries at transaction isolation level 3 on data-only-locked tables)
  - "Range" indicates a range lock (for range queries at transaction isolation level 3 on data-only-locked tables)

  These new values may appear in combination with "Fam dur" (which replaces "Sync pt duration") and with each other, as applicable.

sp_object_stats

Function

Prints lock contention, lock wait-time, and deadlock statistics for tables and indexes.

Syntax

```
sp_object_stats interval [, top_n
   [, dbname, objname [, rpt_option ]]]
```

Parameters

interval – specifies the time period for the sample. It must be in HH:MM:SS form, for example “00:20:00”.

top_n – the number of objects to report, in order of contention. The default is 10.

dbname – the name of the database to report on. If no database name is given, contention on objects in all databases is reported.

objname – the name of a table to report on. If a table name is specified, the database name must also be specified.

rpt_option – must be either **rpt_locks** or **rpt_objlist**.

Examples

1. `sp_object_stats "00:20:00"`

   Reports lock statistics on the top 10 objects server-wide.

2. `sp_object_stats "00:20:00", 5, pubtune`

   Reports only on tables in the *pubtune* database, and lists the five tables that experienced the highest contention.

3. `sp_object_stats "00:15:00", @rpt_option = "rpt_objlist"`

   Prints only the names of the tables that had the highest locking activity, even if contention and deadlocking does not take place.

Comments

- **sp_object_stats** reports on the shared, update, and exclusive locks acquired on tables during a specified sample period. The following reports shows the *titles* tables:

```
Object Name: pubtune..titles (dbid=7, objid=208003772,lockscheme=Datapages)

 Page Locks     SH_PAGE               UP_PAGE               EX_PAGE$
 ----------    ----------            ----------            ----------
 Grants:            94488                 4052                  4828
 Waits:               532                  500                   776
 Deadlocks:             4                    0                    24
 Wait-time:      20603764 ms           14265708 ms           2831556 ms
 Contention:         0.56%                10.98%                13.79%

*** Consider altering pubtune..titles to Datarows locking.
```

- Table 14-1 shows the meaning of the values.

Table 14-1: Output of sp_object_stats

Output Row	Value
Grants	The number of times the lock was granted immediately.
Waits	The number of times the task needing a lock had to wait.
Deadlocks	The number of deadlocks that occurred.
Wait-times	The total number of milliseconds that all tasks spent waiting for a lock.
Contention	The percentage of times that a task had to wait or encountered a deadlock.

- **sp_object_stats** recommends changing the locking scheme when total contention on a table is more than 15 percent, as follows:
  - If the table uses allpages locking, it recommends changing to datapages locking
  - If the table uses datapages locking, it recommends changing to datarows locking.
- *rpt_option* specifies the report type:
  - **rpt_locks** reports grants, waits, deadlocks and wait times for the tables with the highest contention. **rpt_locks** is the default.
  - **rpt_objlist** reports only the names of the objects that had the highest level of lock activity
- **sp_object_stats** creates a table named *tempdb..syslkstats*. This table is not dropped when the stored procedure completes so that it can be queried by a System Administrator.

- Only one user at a time should execute **sp_object_stats**. If more than one user tries to run **sp_object_stats** simultaneously, the second command may be blocked, or the results may be invalid.
- The *tempdb..syslkstats* table is dropped and re-created each time **sp_object_stats** is executed.
- The structure of *tempdb..syslkstats* is described in Table 14-2.

Table 14-2: Columns in the tempdb..syslkstats table

Column Name	Datatype	Description
dbid	*smallint*	Database ID
objid	*int*	Object ID
lockscheme	*smallint*	Integer values 1–3: Allpages = 1, Datapages = 2, Datarows = 3
page_type	*smallint*	Data page = 0, or index page = 1
stat_name	*char(30)*	The statistics represented by this row
stat_value	*float*	The number of grants, waits or deadlocks, or the total wait time

The values in the *stat_name* column are composed of three parts:

- The first part is “ex” for exclusive lock, “sh” for shared lock, or “up” for update lock.
- The second part is “pg” for page locks, or “row” for row locks.
- The third part is “grants” for locks granted immediately, “waits” for locks that had to wait for other locks to be released, “deadlocks” for deadlocks, and “waittime” for the time waited to acquire the lock.

- If you specify a table name, **sp_object_stats** displays all tables by that name. If more than one user owns a table with the specified name, output for these tables displays the object ID, but not the owner name.

Permission

Only a System Administrator can execute **sp_object_stats**.

Tables Used

Creates the table *tempdb..syslkstats*. This table is not dropped at the end of execution and can be queried via Transact-SQL.

See Also

Commands	**alter table**

sp_setrowlockpromote

Function

Sets or changes row-lock promotion thresholds for a datarows-locked table, for all datarows-locked tables in a database, or for all datarows-locked tables on a server.

Syntax

```
sp_setrowlockpromote "server", NULL, new_lwm,
   new_hwm, new_pct
sp_setrowlockpromote {"database" | "table"}, objname,
   new_lwm, new_hwm, new_pct
```

Parameters

server – sets server-wide values for the row lock promotion thresholds.

"database" | **"table"** – specifies whether to set the row-lock promotion thresholds for a database or table.

objname – is either the name of the table or database for which you are setting the row-lock promotion thresholds or **null**, if you are setting server-wide values.

new_lwm – specifies the value to set for the low watermark (LWM) threshold. The LWM must be less than or equal to the high watermark (HWM). The minimum value for LWM is 2. This parameter can be **null**.

new_hwm – specifies the value to set for the high watermark (HWM) threshold. The HWM must be greater than or equal to the LWM. The maximum HWM is 2,147,483,647. This parameter can be **null**.

new_pct – specifies the value to set for the lock promotion percentage (PCT) threshold. PCT must be between 1 and 100. This parameter can be **null**.

Examples

1. `sp_setrowlockpromote "database", engdb, 400, 400,95`

   Sets row lock promotion values for all datarows-locked tables in the *engdb* database.

2. `sp_setrowlockpromote "table", sales, 250, 250, 100`

Sets row lock promotion values for the *sales* table.

Comments

- **sp_setrowlockpromote** sets or changes row-lock promotion thresholds for a table, a database, or Adaptive Server.

  Adaptive Server acquires row locks on a datarows-locked table until the number of locks exceeds the lock promotion threshold. If Adaptive Server is successful in acquiring a table lock, the row locks are released.

  When the number of row locks on a table exceeds the HWM, Adaptive Server attempts to escalate to a table lock. When the number of row locks on a table is below the LWM, Adaptive Server does not attempt to escalate to a table lock. When the number of row locks on a table is between the HWM and LWM, and the number of row locks exceeds the PCT threshold as a percentage of the number of rows in a table, Adaptive Server attempts to escalate to a table lock.
- Lock promotion is always two-tiered, that is, row locks are promoted to table locks. Adaptive Server does not promote from row locks to page locks.
- Lock promotion thresholds for a table override the database or server-wide settings. Lock promotion thresholds for a database override the server-wide settings.
- To change the lock promotion thresholds for a database, you must be using the *master* database. To change the lock promotion thresholds for a table in a database, you must be using the database where the table resides.
- Server-wide row lock promotion thresholds can also be set with **sp_configure**. When you use **sp_setrowlockpromote** to change the values server-wide, it changes the configuration parameters, and saves the configuration file. When you first install Adaptive Server, the server-wide row lock promotion thresholds set by the configuration parameters are:

row lock promotion HWM	200
row lock promotion LWM	200
row lock promotion PCT	100

  See Chapter 16, “New and Changed Configuration Parameters,” for more information.

- The system procedure **sp_sysmon** reports on row lock promotions. See “Lock Management” on page 20-1 for more information.
- Database-level row lock promotion thresholds are stored in the *master..sysattributes* table. If you dump a database, and load it only another server, you must set the row lock promotion thresholds on the new server. Object-level row lock promotion thresholds are stored in the *sysattributes* table in the user database, and are included in the dump.

Permissions

Only a System Administrator can execute **sp_setrowlockpromote**.

Tables Used

master.dbo.sysattributes, master.dbo.sysconfigures, sysattributes

See Also

System procedures	**sp_droprowlockpromote**

15 *optdiag* Utility

This chapter explains how to use the **optdiag** utility.

For information on interpreting and using the results of the **optdiag** utility, see “Viewing Statistics with the optdiag Utility” on page 8-6.

optdiag

Function

Displays optimizer statistics or loads updated statistics into system tables.

Syntax

```
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] [-Tflag_value]
   [-z language] [-J client_charset]
   [-a display_charset]
```

Parameters

binary – extracts statistics in human-readable form and in binary form. Loads binary statistics into system tables when **optdiag** is used with an input file.

simulate – specifies that **optdiag** should display or load simulated statistics. See "Using Simulated Statistics" on page 8-29.

-i *input_file* – specifies the name of the operating system file to use for **optdiag** input. Specifying an input file updates optimizer statistics for the table or column, using the values in the specified file.

database – is the name of the database. In input mode, the database name in the file is used, and the database cannot be specified on the command line.

owner – is the name of a table owner. In display mode, if you do not specify an owner, **optdiag** displays output for all owners of a table if a table name is specified. In input mode, the table owner specified on the command line is ignored, and the value in the input file is used.

table – is the name of the table. If no owner name or table name is provided, statistics for all tables in the database are displayed. If an owner name is provided, but no table name, all tables for the specified owner are displayed. In input mode, the table name specified on the command line is ignored, and the value from the input file is used.

column – is the name of a column. If no column name is provided, all statistics for a table are displayed. In input mode, the column name on the command line is ignored, and the values from the input file are used.

-o *output_file* – specifies the name of an operating system file to store the output from **optdiag**. Any existing file of the same name is overwritten without warning.

-U *username* – specifies an Adaptive Server login name.

-P *password* – specifies your Adaptive Server password. If you do not specify the **-P** flag, **optdiag** prompts for a password.

-I *interfaces_file* – specifies the name and location of the interfaces file to use when connecting to Adaptive Server. If you do not specify **-I** and an interfaces file name, **optdiag** looks for a file named *interfaces* in the directory specified by the SYBASE environment variable.

-S *server* – specifies the name of the Adaptive Server to which to connect. **optdiag** looks for this name in the interfaces file. If you specify **-S** with no argument, **optdiag** looks for a server named SYBASE. If you do not specify **-S**, **optdiag** looks for the server specified by the DSQUERY environment variable.

-v – displays the version number of **optdiag** and a copyright message and returns to the operating system

-h – displays the **optdiag** syntax help.

-T*flag_value* – sets trace flags for the **optdiag** session. The **optdiag** trace flags are shown in Table 15-1.

Table 15-1: optdiag trace flag values

Flag Value	Meaning
1	Do not stop with a warning if the **optdiag** version of Adaptive Server being used does not match the Adaptive Server version in input file.
2	Print status messages "Next table is *table_name*" in input mode.
4	Skip consistency checking for step numbers while loading histograms in input mode.
6	Display lines of input file during input mode. This flag has no effect in display mode

Examples

1. **`optdiag statistics pubtune -Usa -Ppasswd -o pubtune.opt`**

   Displays statistics for all tables in the *pubtune* database, placing the output in the *pubtune.opt* file.

2. **`optdiag statistics pubtune..titles -Usa -Ppasswd -o titles.opt`**

   Displays statistics for the *titles* table.

3. **`optdiag statistics pubtune..titles -Usa -Ppasswd -o titles.opt -J roman8 -z french`**

   Displays statistics using the roman8 character set. Row labels and error messages are displayed in French.

4. **`optdiag binary statistics pubtune..titles.price -Usa -Ppasswd -o price.opt`**

   Displays binary statistics for the *price* column in the *titles* table.

5. **`optdiag statistics -i price.opt -Usa -Ppasswd`**

   Loads edited statistics from the *price.opt* file.

Comments

- See Chapter 8, "Statistics Enhancements," for an explanation of **optdiag** output.

- When you use **binary** mode, the human-readable values are printed with comment marks, "#", at the beginning of the lines, as shown in this example:

```
Statistics for column:              "price"
Last update of column statistics:   Jan 20 1998  7:16PM
Statistics loaded from Optdiag.

     Range cell density:            0x3f8b9cfefece26bf
#    Range cell density:            0.0134830400000000
     Total density:                 0x3f8b9cfefece26bf
#    Total density:                 0.0134830400000000
     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)
```

- When using **optdiag** with an input file to change statistics, all characters after the "#" in a line are ignored.
- Converting floating-point values may lead to rounding errors when files are used for input. Editing statistics using the binary values provides greater precision, as long as you are loading statistics on the same hardware platform.

Byte Ordering and Binary *optdiag* Files

- Do not use the **binary** mode option to move statistics between Adaptive Servers on machines that have different byte ordering. On an incompatible architecture server, always comment out binary statistics and load the human-readable statistics. On a compatible architecture server, either binary statistics or human-readable statistics can be loaded.

Input Mode

- When you use the **-i** *input_file* syntax, **optdiag** reads the file and updates statistics in *sysstatistics*.
- **optdiag** input mode resets the configuration parameter **allow update to system tables** to 1 at the beginning of the session and resets the value to 0 at the end of the session.
- 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** trace 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 to represent the null value.
  - Two adjacent cells must not both use the **<** operator.

- For more information on changing statistics using **optdiag**, see "Changing Statistics with optdiag" on page 8-24.

➤ ***Note***

optdiag works only with single-byte character sets. If your server uses a multibyte character set, **optdiag** prints a warning message and exits.

System Administration

16 New and Changed Configuration Parameters

New Configuration Parameters

The following configuration parameters are new in Adaptive Server 11.9.2:

Configuration Parameter	See
default exp_row_size percent	default exp_row_size percent, page 16-12
enable housekeeper GC	enable housekeeper GC, page 16-13
license information	license information, page 16-14
lock hashtable size	lock hashtable size, page 16-3
lock scheme	lock scheme, page 16-4
lock spinlock ratio	lock spinlock ratio, page 16-4
lock wait period	lock wait period, page 16-5
read committed with lock	read committed with lock, page 16-8
row lock promotion HWM	row lock promotion HWM, page 16-9
row lock promotion LWM	row lock promotion LWM, page 16-10
row lock promotion PCT	row lock promotion PCT, page 16-11

➤ ***Note***

Datarows locking may require changing the value for the **number of locks** configuration parameter. See “Estimating number of locks for Data-Only-Locked Tables” on page 18-18.

Changed Configuration Parameters

This section describes the configuration parameters that are changed in Adaptive Server 11.9.2.

Renamed Configuration Parameters

The following configuration parameters have been renamed:

Old Name	New Name	See
lock promotion HWM	**page lock promotion HWM**	page lock promotion HWM, page 16-6
lock promotion LWM	**page lock promotion LWM**	page lock promotion LWM, page 16-7
lock promotion PCT	**page lock promotion PCT**	page lock promotion PCT, page 16-8

Replaced Configuration Parameters

The new **lock spinlock ratio** parameter replaces the following configuration parameters:

- **address lock spinlock ratio**
- **table lock spinlock ratio**
- **page lock spinlock ratio**

For information, see "lock spinlock ratio" on page 16-4.

➤ ***Note***

If you start Adaptive Server with a configuration file from a pre-11.9 version of the server, the values for the lock promotion and lock spinlock ratio parameters are set to default values.

Default *total memory* Configuration Parameter Value

The default value for the **total memory** configuration parameter has been increased. See the Release Bulletin, or use the following command to see the default value and current setting:

```
sp_configure "total memory"
```

Configuration Parameter Summaries

This section provides summaries of the new and changed configuration parameters. The parameters are listed in alphabetical order under the group names used in the configuration file and **sp_configure** output.

Lock Manager

Use the parameters in this group to configure locks.

lock hashtable size

Summary Information	
Default value	2048
Range of values	1–2147483647
Status	Static
Display level	Comprehensive
Required role	System Administrator

The **lock hashtable size** parameter specifies the number of **hash buckets** in the lock hash table. This table manages all row, page, and table locks and all lock requests. Each time a task acquires a lock, the lock is assigned to a hash bucket, and each lock request for that lock checks the same hash bucket. Setting this value too low results in large numbers of locks in each hash bucket and slows the searches. On Adaptive Servers with multiple engines, setting this value too low can also lead to increased spinlock contention. You should not set the value to less than the default value, 2048.

lock hashtable size must be a power of 2. If the value you specify is not a power of 2, **sp_configure** rounds the value to the next highest power of 2 and prints an informational message.

The optimal hash table size is a function of the number of distinct objects (pages, tables, and rows) that will be locked concurrently. The optimal hash table size is at least 20 percent of the number of distinct objects that need to be locked concurrently. See “Lock Hash Table Information” on page 20-1 for more information on configuring the lock hash table size.

lock scheme

Summary Information	
Default value	allpages
Range of values	allpages, datapages, datarows
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

The **lock scheme** parameter sets the default locking scheme to be used by **create table** and **select into** commands when a lock scheme is not specified in the command.

The values for lock scheme are character data, so you must use 0 as a placeholder for the second parameter, which must be numeric, and specify **allpages**, **datapages**, or **datarows** as the third parameter:

```
sp_configure "lock scheme", 0, datapages
```

lock spinlock ratio

Summary Information	
Default value	85
Range of values	1–2147483647
Status	Static
Display level	Comprehensive
Required role	System Administrator

Adaptive Server manages the acquiring and releasing of locks using an internal hash table with a configurable number of hash buckets. On SMP systems, this hash table can use one or more spinlocks to serialize access between processes running on different engines. To set the number of hash buckets in the lock hash table, use the **lock hashtable size** configuration parameter.

For Adaptive Servers running with multiple engines, the **lock spinlock ratio** sets a ratio that determines the number of lock hash buckets that are protected by one spinlock. If you increase **lock hashtable size**, the

number of spinlocks increases, so the number of hash buckets protected by one spinlock remains the same.

Adaptive Server's default value for **lock spinlock ratio** is 85. With **lock hashtable size** set to the default value of 2048, the default spinlock ratio defines 26 spinlocks for the lock hash table. For more information about configuring spinlock ratios, see "Configuring Spinlock Ratio Parameters," in Chapter 10, "Managing Multiprocessor Servers" in the *System Administration Guide*.

The system procedure **sp_sysmon** reports on the average length of the hash chains in the lock hash table. See "Lock Hash Table Information" on page 20-1.

lock wait period

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

The **lock wait period** limits the number of seconds that tasks wait to acquire a lock on a table, data page, or data row. If the task does not acquire the lock within the specified time period, Adaptive Server returns error message 12205 to the user and rolls back the transaction.

The **lock wait** option of the **set** command sets a session-level number of seconds that a task will wait for a lock. It overrides the server-level setting for the session.

At the default value, all processes wait indefinitely for locks. To restore the default value, if **lock wait period** has been changed, reset the value to 2147483647, or use the command:

```
sp_configure "lock wait period", 0, "default"
```

page lock promotion HWM

Summary Information	
Default value	200
Range of values	2–2147483647
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

The **page lock promotion HWM** (high water mark) parameter, together with the **page lock promotion LWM** (low water mark) and **page lock promotion PCT** (percentage) configuration parameters, specify the number of page locks permitted during a single scan session of a page-locked table or index before Adaptive Server attempts to escalate from page locks to a table lock.

The **page lock promotion HWM** parameter sets a maximum number of page locks allowed on a table before Adaptive Server attempts to escalate to a table lock. When the number of page locks acquired during a scan session exceeds **page lock promotion HWM**, Adaptive Server attempts to acquire a table lock. The **page lock promotion HWM** value cannot be higher than **number of locks** value.

For more detailed information on scan sessions and setting up page lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

The default value (200) for **page lock promotion HWM** is appropriate for most applications. You might want to raise the value to avoid table locking. For example, if you know that there are regular updates to 500 pages of an allpages-locked or datapages-locked table with containing thousands of pages, you can increase concurrency for the tables by setting **page lock promotion HWM** to around 500 so that lock promotion does not occur at the default setting of 200.

You can also configure lock promotion of page-locked tables and views at the per-object level. See "sp_setrowlockpromote" on page 14-23.

Use **sp_sysmon** to see how changing the **page lock promotion HWM** parameter affects the number of lock promotions. **sp_sysmon** reports the number of exclusive page to exclusive table lock promotions and the number of shared page to shared table lock promotions. See

"Lock Promotions" in Chapter 24, "Monitoring Performance with sp_sysmon," in the *Performance and Tuning Guide*.

page lock promotion LWM

Summary Information	
Default value	200
Range of values	2–value of **page lock promotion HWM**
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

The **page lock promotion LWM** low water mark) parameter, together with the **page lock promotion HWM** (high water mark) and the **page lock promotion PCT** (percentage), specify the number of page locks permitted during a single scan session of a page locked table or an index before Adaptive Server attempts to promote from page locks to a table lock.

The **page lock promotion LWM** sets the number of page locks below which Adaptive Server does not attempt to issue a table lock on an object. The **page lock promotion LWM** parameter must be less than or equal to **page lock promotion HWM**.

For more information on scan sessions and setting up lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

The default value (200) for **page lock promotion LWM** is sufficient for most applications. If Adaptive Server runs out of locks (except for an isolated incident), you should increase **number of locks**. See "Estimating number of locks for Data-Only-Locked Tables" on page 18-18.

You can also configure page lock promotion at the per-object level. See **sp_setpglockpromote** in the *Adaptive Server Reference Manual*.

page lock promotion PCT

Summary Information	
Default value	100
Range of values	1–100
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

If the number of locks held on an object is between **page lock promotion LWM** (low water mark) and **page lock promotion HWM** (high water mark), the **page lock promotion PCT** (percentage) parameter sets the percentage of page locks (based on the table size) above which Adaptive Server attempts to acquire a table lock.

For more detailed information on setting up page lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

The default value (100) for **page lock promotion PCT** is appropriate for most applications.

You can also configure lock promotion at the per-object level for page locked objects. See **sp_setpglockpromote** in the *Adaptive Server Reference Manual*.

read committed with lock

Summary Information	
Default value	0 (off)
Valid values	0 (off), 1(on)
Status	Dynamic
Display level	Comprehensive
Required role	System Administrator

The **read committed with lock** parameter determines whether an Adaptive Server using transaction isolation level 1 (read committed) holds shared locks on rows or pages of data-only locked tables

during **select** queries. For cursors, the option applies only to cursors declared as read-only cursors. By default, this parameter turned off (set to 0) in order to reduce lock contention and blocking. This parameter affects only queries on data-only locked tables.

For transaction isolation level 1, **select** queries on allpages-locked tables continue to hold locks on the page at the current position. Any implicitly or explicitly updatable cursor on a data-only locked table also holds locks on the current page or row. See "Locking for select Queries at Isolation Level 1" on page 5-5 for more information.

row lock promotion HWM

Summary Information	
Default value	200
Range of values	2–2147483647
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

The **row lock promotion HWM** (high water mark) parameter, together with **row lock promotion LWM** (low water mark) and **row lock promotion PCT** (percentage) specify the number of row locks permitted during a single scan session of a table or an index before Adaptive Server attempts to escalate from row locks to a table lock.

The **row lock promotion HWM** parameter sets a maximum number of row locks allowed on a table before Adaptive Server attempts to escalate to a table lock. When the number of locks acquired during a scan session exceeds **row lock promotion HWM**, Adaptive Server attempts to acquire a table lock. The **lock promotion HWM** value cannot be higher than the **number of locks** value.

For more information on scan sessions and setting up lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

The default value (200) for **row lock promotion HWM** is appropriate for most applications. You might want to raise the value to avoid table locking. For example, if you know that there are regular updates to 500 rows on a table that has thousands of rows, you can increase

concurrency for the tables by setting **row lock promotion HWM** to around 500.

You can also configure row lock promotion at the per-object level. See "sp_setrowlockpromote" on page 14-23.

row lock promotion LWM

Summary Information	
Default value	200
Range of values	2–value of **row lock promotion HWM**
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

The **row lock promotion LWM** (low water mark) parameter, together with the **row lock promotion HWM** (high water mark) and **row lock promotion PCT** (percentage), specify the number of row locks permitted during a single scan session of a table or an index before Adaptive Server attempts to promote from row locks to a table lock.

The **row lock promotion LWM** parameter sets the number of locks below which Adaptive Server does not attempt to acquire a table lock on the object. The **row lock promotion LWM** parameter must be less than or equal to **row lock promotion HWM**.

For more detailed information on scan sessions and setting up lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5 "Locking in Adaptive Server" in the *Performance and Tuning Guide*.

The default value (200) for **row lock promotion LWM** is sufficient for most applications. If Adaptive Server runs out of locks (except for an isolated incident), you should increase **number of locks**. See "Estimating number of locks for Data-Only-Locked Tables" on page 18-18.

You can also configure lock promotion at the per-object level. See "sp_setrowlockpromote" on page 14-23.

row lock promotion PCT

Summary Information	
Default value	100
Range of values	1–100
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

If the number of locks held on an object is between **row lock promotion LWM** (low water mark) and **row lock promotion HWM** (high water mark), the **row lock promotion PCT** (percentage) parameter sets the percentage of row locks (based on the number of rows in the table) above which Adaptive Server attempts to acquire a table lock.

For more information on setting up lock promotion limits, see "Configuring Locks and Lock Promotion Thresholds," in Chapter 5, "Locking in Adaptive Server," in the *Performance and Tuning Guide*.

The default value (100) for **row lock promotion PCT** is appropriate for most applications.

You can also configure row lock promotion at the per-object level. See "sp_setrowlockpromote" on page 14-23.

SQL Server Administration

Use the parameters in this group to maintain Adaptive Server.

default exp_row_size percent

Summary Information	
Default value	5
Range of values	0–100
Status	Dynamic
Display level	Intermediate
Required role	System Administrator

The **default exp_row_size percent** parameter reserves space for expanding updates in data-only-locked tables, in order to reduce row forwarding. An **expanding update** is any update to a data row that increases the length of the row. Data rows that allow null values or that have variable-length columns may be subject to expanding updates. In data-only-locked tables, expanding updates can require row forwarding if the data row increases in size so that it no longer fits on the page.

The default value, 5, sets aside 5 percent of the available data page size for use by expanding updates. Since 2002 bytes are available for data storage on pages in data-only-locked tables, this leaves 100 bytes for expansion. This value is only applied to pages for tables that have variable-length columns.

Valid values are 0–99. Setting **default exp_row_size percent** to 0 means that all pages are completely filled and no space is left for expanding updates.

default exp_row_size percent is applied to data-only-locked tables with variable-length columns when **exp_row_size** is not explicitly provided with **create table** or set with **sp_chgattribute**. If a value is provided with **create table**, that value takes precedence over the configuration parameter setting. See “Reducing Row Forwarding with Expected Row Size” on page 4-1 for more information.

enable housekeeper GC

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

When **enable housekeeper GC** is set to 1, the default value, the housekeeper task performs space reclamation on data-only-locked tables. When a user task deletes a row from a data-only-locked table, a task is queued to the housekeeper to check the data and index pages for committed deletes.

When **enable housekeeper GC** is set to 0, the housekeeper does not perform space reclamation. Use this setting:

- If you use only allpages locking
- If there are few deletes performed on your data-only-locked tables
- If your workload leaves little idle CPU time

The system procedure sp_sysmon reports on how often the housekeeper task performed space reclamation and how many pages were reclaimed. See “Housekeeper Task Activity” on page 20-6.

license information

Summary Information	
Default value	0
Valid values	$0–2^{31}$
Status	Dynamic
Display level	10
Required role	System Administrator

The **license information** parameter allows Sybase System Administrators to monitor the number of user licenses used in Adaptive Server. Enabling license monitoring only monitors the number of licenses issued; it does not enforce the license agreement.

If **license information** is set to 0, Adaptive Server does not monitor license use. If **license information** is set to a number greater than 0, the housekeeper task monitors the number of licenses used during the idle cycles in Adaptive Server. Set **license information** to the number of licenses specified in your license agreement.

If the number of licenses used is greater than the number to which **license information** is set, Adaptive Server writes the following error message to the error log:

```
WARNING: Exceeded configured number of user licenses
```

At the end each 24-hour period, the maximum number of licenses used during that time is added to the *syblicenseslog* table. The 24 hour period restarts if Adaptive Server is restarted.

See “Using the License Use Monitor” on page 18-19 for more information.

17 Using the *reorg* Command

Update activity against a table can eventually lead to inefficient utilization of space and reduced performance. The **reorg** command reorganizes the use of table space and improves performance.

This chapter discusses the following topics:

reorg Subcommands

The **reorg** command provides four subcommands for carrying out different types and levels of reorganization:

- **reorg forwarded_rows** undoes row forwarding.
- **reorg reclaim_space** reclaims unused space left on a page as a result of deletions and row-shortening updates.
- **reorg compact** both reclaims space and undoes row forwarding.
- **reorg rebuild** undoes row forwarding and reclaims unused page space, as does **reorg compact.** In addition, **reorg rebuild**:
  - Rewrites all rows to accord with a table's clustered index, if it has one
  - Writes rows to data pages to accord with any changes made in space management settings through **sp_chgattribute** (see "sp_chgattribute" on page 14-2).
  - Drops and re-creates all indexes belonging to the table

The **reclaim_space**, **forwarded_rows**, and **compact** subcommands:

- Minimize interference with other activities by using multiple small transactions of brief duration. Each transaction is limited to eight pages of **reorg** processing.
- Provide **resume** and **time** options that allow you to set a time limit on how long a **reorg** runs and to resume a **reorg** from the point at which the previous **reorg** stopped. This makes it possible to use a series of partial reorganizations at off-peak times to **reorg** a large table. For information on the **resume** and **time** options, see "resume and time Options for Reorganizing Large Tables" on page 17-8.

The following considerations apply to the **rebuild** subcommand:

- **reorg rebuild** holds an exclusive table lock for its entire duration. On a large table this may be a significant amount of time. However, **reorg rebuild** accomplishes everything that dropping and re-creating a clustered index does and takes less time. In addition, **reorg rebuild** rebuilds the table using all of the table's current space management settings. Dropping and re-creating an index does not use the space management setting for **reservepagegap**.
- In most cases, **reorg rebuild** requires additional disk space equal to the size of the table it is rebuilding and its indexes.

Command Syntax

The command syntax for **reorg** is:

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]
reorg forwarded_rows tablename
   [with {resume, time = no_of_minutes}]
reorg compact tablename
   [with {resume, time = no_of_minutes}]
reorg rebuild tablename
```

The following restrictions hold:

- The table specified in the command must use either the **datarows** or **datapages** locking scheme.
- You must be a System Administrator or the object owner to issue the **reorg** command.
- You cannot issue the **reorg** command within a transaction.

When to Run a *reorg* Command

The **reorg** command is useful when:

- A large number of forwarded rows causes extra I/O during read operations.
- Inserts and serializable reads are slow because they encounter pages with noncontiguous free space that needs to be reclaimed.
- Large I/O operations are slow because of low cluster ratios for data and index pages.
- **sp_chgattribute** was used to change a space management setting (**reservepagegap**, **fillfactor**, or **exp_row_size**) and the change is to be applied to all existing rows and pages in a table, not just to future updates.

Using the *optdiag* Utility to Assess the Need for a *reorg*

To assess the need for running a **reorg,** you can use statistics from the *systabstats* table and the **optdiag** utility. The *systabstats* table contains statistics on the utilization of table space. The **optdiag** utility generates reports based on statistics in the *systabstats* table and the *sysstatistics* table.

For information on the *systabstats* table and the **optdiag** utility, see Chapter 8, “Statistics Enhancements,” and Chapter 15, “optdiag Utility.”

Information available through the **optdiag** utility includes:

- Number of data pages in a table.
- Number of pages that have only deleted rows.
- Number of data rows in a table.
- Number of forwarded rows.
- Number of deleted rows that belong to committed transactions, but whose space has not been reclaimed.
- Cluster ratios for data pages, index pages, and data rows. Cluster ratios can be used to evaluate the effectiveness of large I/O for scans.

Space Reclamation Without the *reorg* Command

Several types of activities reclaim or reorganize the use of space in a table on a page-by-page basis:

- Inserts, when an insert encounters a page that would have enough room if it reclaimed unused space.
- The **update statistics** command (for index pages only)
- Re-creating clustered indexes
- The housekeeper task, if **enable housekeeper GC** is set to **1**

Each of these activities has its limitations and may be insufficient if space on a large number of pages needs to be reclaimed or reorganized. For example, inserts may execute more slowly when they need to reclaim space and inserts may not affect many pages with space that can be reorganized. Space reclamation under the housekeeper task compacts unused space, but it runs only when no other tasks are requesting CPU time, so it may not reach every page that needs it.

Moving Forwarded Rows to Home Pages

If an update makes a row too long to fit on its current page, the row is forwarded to another page. A reference to the row is maintained on its original page, the row's **home** page, and all access to the forwarded row goes through this reference. Thus, it always takes two page accesses to get to a forwarded row. If a scan needs to read a large number of forwarded pages, the I/Os caused by extra page accesses slow performance.

The **reorg forwarded_rows** command undoes row forwarding by either moving a forwarded row back to its home page, if there is enough space, or by deleting the row and reinserting it in a new home page.

You can get statistics on the number of forwarded rows in a table by querying the *systabstats* table and by using the **optdiag** utility.

You can undo row forwarding with the **reorg rebuild**, **reorg compact**, or **reorg forwarded_rows** command:

- Use **reorg forwarded_rows** when rebuilding an entire table (**reorg rebuild**) is not necessary and unused space from deletions and updates is not a problem.
- Use **reorg compact** if you need to undo row forwarding as well as reclaim unused space. In some cases, you may also need to run

reorg compact to remove row forwarding, even if there is no need to run it for reclaiming unused space.

reorg forwarded_rows Syntax

The syntax for **reorg forwarded_rows** is:

```
reorg forwarded_rows tablename
   [with {resume, time = no_of_minutes}]
```

For information about the **resume** and **time** options, see "resume and time Options for Reorganizing Large Tables" on page 17-8.

reorg forwarded_rows does not apply to indexes, because indexes do not have forwarded rows.

Using *reorg compact* to Remove Row Forwarding

reorg forwarded_rows uses allocation page hints to find forwarded rows. This makes it fast, since it does not have to search an entire table, but it also means that it may miss some forwarded rows. After running **reorg forwarded_rows**, you can evaluate its effectiveness by using the **optdiag** utility and checking the value given for "Forwarded row count." If "Forwarded row count" is high, you can then run **reorg compact**, which goes through a table page by page and undoes all row forwarding.

Reclaiming Unused Space from Deletes and Updates

When a task performs a delete operation or an update that shortens row length, the empty space is not reclaimed at the time of the transaction—the space is preserved in case the transaction is rolled back. If a table is subject to frequent deletes and row-shortening updates, unreclaimed space may accumulate to the point that it impairs performance.

The **reorg reclaim_space** command reclaims unused space left by deletes and updates. On each page that has space resulting from committed deletes or row-shortening updates, **reorg reclaim_space** rewrites the remaining rows contiguously, leaving all the unused space at the end of the page. If all rows have been deleted and there are no remaining rows, **reorg reclaim_space** deallocates the page.

You can get statistics on the number of unreclaimed row deletions in a table from the *systabstats* table and by using the **optdiag** utility. There

is no direct measure of how much unused space there is as a result of row-shortening updates.

You can reclaim unused table space resulting from deletions and updates by using the **reorg rebuild**, **reorg compact**, or **reorg reclaim_space** command:

- Use **reorg reclaim_space** when there is no need to fully rebuild a table (**reorg rebuild**) and excessive row forwarding is not a concern.
- Use **reorg compact** if you need to reclaim unused space and undo row forwarding.

reorg reclaim_space Syntax

The syntax for **reorg reclaim_space** is:

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]
```

If you specify only a table name, the table's data pages are reorganized to reclaim unused space. Indexes are not affected. If you specify an index name, only the pages of the index are reorganized.

For information about the **resume** and **time** options, see "resume and time Options for Reorganizing Large Tables" on page 17-8.

Reclaiming Unused Space and Undoing Row Forwarding

The **reorg compact** command combines the functions of the **reorg reclaim_space** and **reorg forwarded_rows** commands. Use **reorg compact** when:

- Rebuilding an entire table (**reorg rebuild**) is not necessary and both row forwarding and unused space from deletes and updates may be affecting performance.
- You have run **forwarded_rows** and find that there are still a large number of forwarded rows. See "Using reorg compact to Remove Row Forwarding" on page 17-5.

reorg compact Syntax

The syntax for **reorg compact** is:

```
reorg compact tablename
   [with {resume, time = no_of_minutes}]
```

For information about the **resume** and **time** options, see "resume and time Options for Reorganizing Large Tables" on page 17-8.

Rebuilding a Table

Use the **reorg rebuild** command when:

- Large I/O is not being selected for queries where it is usually used, and the **optdiag** utility shows a low cluster ratio for datapages, data rows, or index pages. See "Viewing Statistics with the optdiag Utility" on page 8-6.
- You used **sp_chgattribute** to change one or more of the **exp_row_size**, **reservepagegap**, or **fillfactor** space management settings and you want the changes to apply not only to future data, but also to existing rows and pages. For information about **sp_chgattribute**, see "sp_chgattribute" on page 14-2.

If a table needs to be rebuilt because of a low cluster ratio, it may also need to have its space management settings changed (see "Changing Space Management Settings Before Using reorg rebuild" on page 17-8).

The **reorg rebuild** command uses a table's current space management settings to rewrite the rows in the table according to the table's clustered index, if it has one. All indexes on the table are dropped and re-created using the current space management values for **reservepagegap** and **fillfactor**. After a **rebuild**, a table has no forwarded rows and no unused space from deletions or updates.

reorg rebuild Syntax

The syntax for **reorg rebuild** is:

```
reorg rebuild tablename
```

Prerequisites for Running *reorg rebuild*

Before you run **reorg rebuild** on a table:

- Set the database option **select into/bulkcopy/pllsort** to **true** and run **checkpoint** in the database.
- Make sure that additional disk space, equal to the size of the table and its indexes, is available.

To set **select into/bulkcopy/pllsort** to **true** and checkpoint the database, you can use the following **isql** commands:

```
1> use master
2> go
1> sp_dboption pubs2,
    "select into/bulkcopy/pllsort", true
2> go
1> use pubs2
2> go
1> checkpoint
2> go
```

Following a **rebuild** on a table:

- You must dump the database containing the table before you can dump the transaction log
- Distribution statistics for the table are updated
- All stored procedures that reference the table will be recompiled the next time they are run

Changing Space Management Settings Before Using *reorg rebuild*

When **reorg rebuild** rebuilds a table, it rewrites all table and index rows according to the table's current settings for **reservepagegap**, **fillfactor**, and **exp_row_size**. All of these properties have an effect on how rapidly inserts cause a table to become fragmented, as measured by a low cluster ratio.

If it appears that a table quickly becomes fragmented and needs to be rebuilt too frequently, it may be a sign that the table's space management settings need to be changed before you run **reorg rebuild**.

To change the space management settings, use the system procedure **sp_chgattribute** (see "sp_chgattribute" on page 14-2). For information on space management settings, see Chapter 4, "Setting Space Management Properties."

resume and *time* Options for Reorganizing Large Tables

Use the **resume** and **time** options of the **reorg** command when reorganizing an entire table would take too long and would interfere with other database activities. The **time** option allows you to run a **reorg** for a specified length of time. The **resume** option allows you to start a **reorg** at the point in a table where the previous **reorg** left off. In combination, the two options allow you to reorganize a large table by running a series of partial reorganizations (for example, during off-hours), with each **reorg** starting where the previous **reorg** left off.

The **resume** and **time** options are available with the **reclaim_space**, **forwarded_rows**, and **compact** subcommands. They are not available with **reorg rebuild**.

Syntax for Using *resume* and *time* in *reorg* Commands

The syntax for using the **resume** and **time** options with the subcommands that support them is as follows:

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]
reorg forwarded_rows tablename
   [with {resume, time = no_of_minutes}]
reorg compact tablename
   [with {resume, time = no_of_minutes}]
```

The following considerations apply:

- If you specify only the **resume** option, the **reorg** begins at the point where the previous **reorg** stopped and continues to the end of the table.
- If you specify only the **time** option, the **reorg** starts at the beginning of the table and continues for the specified number of minutes.
- If you specify both options, the **reorg** starts at the point where the previous **reorg** left off and continues for the specified number of minutes.

Specifying *no_of_minutes* in the *time* Option

The *no_of_minutes* argument in the **time** option is specified as an integer and refers to elapsed time, not CPU time. For example, to run **reorg compact** for 30 minutes, beginning where a previous **reorg compact** left off, enter:

```
reorg compact tablename with resume, time=30
```

If the **reorg** process goes to sleep during any part of the 30 minutes, it still counts as part of the elapsed time and does not add to the duration of the **reorg**.

When the amount of time specified in *no_of_minutes* has passed, **reorg** saves statistics about the portion of the table or index that was processed in the *systabstats* table. This information is used as the restart point for a **reorg** with the **resume** option. The restart points for each of the three subcommands that take **resume** and **time** options are

maintained separately. You cannot, for example, start a reorg with **reorg reclaim_space** and then resume it with **reorg compact**.

If *no_of_minutes* is specified, and **reorg** arrives at the end of a table or an index before the time is up, it returns to the beginning of the object and continues until it reaches its time limit.

➤ ***Note***

The **resume** and **time** options make it possible to reorganize an entire table or index over multiple runs. However, if there are updates between **reorg** runs, some pages may be processed twice and some pages may not be processed at all.

18 System Administration Changes

Version 11.9.2 includes changes in the way transactions are logged and in the way log space is monitored and reserved. This chapter includes the following topics related to these and other changes:

New Log Space Requirements

In version 11.9.2, the size of the transaction log may need to be increased to make room for newly introduced **rollback** records. Rollback records are logged whenever a transaction is rolled back. For every update record that is rolled back, a rollback record is logged.

To make sure that there is always space for rollback records in the event of a rollback, version 11.9.2 saves enough space to log a rollback record for every update belonging to an open transaction. If a transaction completes successfully, no rollback records are logged and the space reserved for them is released.

To calculate the increased amount of space that needs to be added to an 11.9.2 transaction log to accommodate rollback records, you need to estimate:

- The number of update records in the transaction log that are likely to belong to already rolled-back transactions.
- The maximum number of update records in the transaction log that are likely to belong to open transactions at any one time.

Each rollback record requires approximately 60 bytes of space, or 3 one hundredths of a page. Thus, the calculation for including rollback records (RRs) in the transaction log is:

Added space, in pages = (logged RRs + # open updates) * 3 / 100

In addition to increasing the size of the transaction log to handle rollback records, it may also be useful to add log space to compensate for the effects of rollback records on the last-chance threshold (LCT) and on user-defined thresholds, as described in the following sections.

Effect of Rollback Records on the Last-Chance Threshold

If the transaction log fills up, all update activity in a database is brought to a halt until new log space becomes available. To prevent the log from completely filling up, the Adaptive Server defines a last chance threshold (LCT) in every transaction log. When the LCT is reached, the server generates a message notifying the user that the log is almost full and that:

```
All future modifications to this database will be
suspended until the log is successfully dumped and
space becomes available.
```

Dumping the log copies it to an external device and clears space in the log up to the page immediately before the oldest active transaction. The LCT is defined to be large enough to meet expected logging needs as the dump proceeds and new records are written to the transaction log.

In version 11.9.2, the amount of space reserved by the LCT has been increased to provide room for rollback records. In addition, the LCT is likely to be reached sooner because of the space used by already logged rollback records and the space reserved against open

transactions for potential rollback records. Figure 18-1 illustrates these changes.

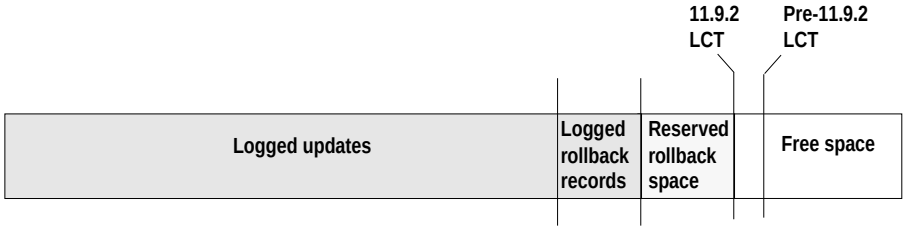

Figure 18-1: Comparing 11.9.2 and pre-11.9.2 LCTs.

In version 11.9.2, the LCT reserves slightly more free space than the pre-11.9.2 LCT. This means that even if there were no other difference, the LCT in 11.9.2 would be reached a little earlier than a pre-11.9.2 LCT. However, rollback records constitute an additional difference.

In Figure 18-1, log space is occupied by logged rollback records for closed transactions that did not complete successfully. In addition, space is reserved for rollback records that may need to be logged, if any of the currently open transactions do not complete successfully. Together, the space for logged rollback records and for potential rollback records is likely to be considerably greater than the extra space for the LCT. Thus, in total, the changes in rollback logging and the LCT mean that the 11.9.2 LCT may be reached significantly sooner than in pre-11.9.2 versions, if the size of the transaction log is not increased.

In general, about 18 percent more log space is reserved for the LCT in 11.9.2 than in earlier versions. For example, for a transaction log of 5000 pages, version 11.5 reserves 264 pages and version 11.9.2 reserves 312 pages. This is a difference of 48 pages and an increase of 18.18 percent from 11.5 to 11.9.2.

Effect of Upgrading to Version 11.9.2 on User-Defined Thresholds

The effect of upgrading to version 11.9.2 on user-defined thresholds is similar to the effect on LCTs. Because of the log space used or reserved for rollback records, a user-defined threshold is reached

sooner, in 11.9.2, than in pre-11.9.2 versions. Figure 18-2 illustrates the effect of upgrading to 11.9.2 on a user-defined threshold.

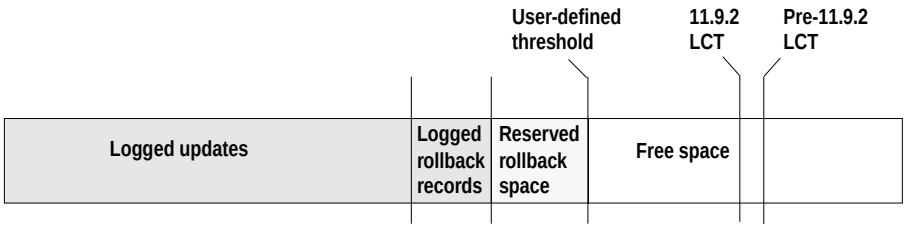

Figure 18-2: Effect of upgrading to version 11.9.2 on user-defined thresholds

A user-defined threshold such as the one in Figure 18-2 is often used to initiate a **dump transaction.** The threshold is set so that there is enough room to complete the dump before the LCT is reached and all open transactions in the log are suspended. In 11.9.2, the LCT is moved slightly, so there is less free space after the user-defined threshold for completing a dump than in a pre-11.9.2 version. However, the loss of space for a dump because of the increased LCT is likely to be more than compensated for by the space reserved for rollback records for open transactions.

LCT and User Log Caches for Shared Log and Data Segments

In pre11.9.2 versions, because of the way Adaptive Server monitored and reserved log space, a database with shared log and data segments could not have an LCT and did not allow transactions to be buffered in a user log cache. In version 11.9.2, both restrictions are removed. Every database now has a last-chance threshold and all databases allow transactions to be buffered in a user log cache.

In version 11.9.2, when a database with shared log and data segments is created, an initial LCT is assigned, based on the size of the user's model database. Subsequently, the size of the LCT is recalculated dynamically, based on the amount and kind of activity in the database.

You can get the current size of the LCT for a database with shared log and data segments using the **lct_admin** system function as follows:

```
select lct_admin("reserve",0)
```

This use of **lct_admin reserve** is described further under "Using lct_admin reserve to Get the Current Last-Chance Threshold" on page 18-11.

Reaching LCT Suspends Transactions

In pre-11.9.2 versions, when no space is left for logging transactions in a database with shared log and data segments, the server generates an error message, and all open transactions are terminated. In 11.9.2, the default behavior is to suspend open transactions until additional log space is created. Transactions suspended because of the LCT can be terminated using the new **abort** parameter of the **lct_admin** system function (see "lct_admin abort" on page 18-8).

Using *abort tran on log full* to Abort Transactions

To have the LCT automatically abort open transactions, rather than suspend them, use the **abort tran on log full** option with **sp_dboption:**

```
sp_dboption database_name "abort tran on log full",
   true
```

If you upgrade to version 11.9.2 from an earlier version in which the **abort tran on log full** option is set, **abort tran on log full** is carried over to 11.9.2.

LCT Assigned for Shared Log and Data Segments

When a database with shared log and data segments is upgraded to version 11.9.2, it is automatically assigned a last-chance threshold.

Using *alter database* When the Master Database Reaches the LCT

In pre-11.9.2 versions, the *master* database does not have an LCT, because it has shared log and data segments. In version 11.9.2, the *master* database does have an LCT.

When the LCT on the *master* database is reached, you can use **alter database** to add space to the *master* database's transaction log. This allows suspended transactions in the log to become active and allows further activity in the server. However, while the master transaction log is at its LCT, you cannot use **alter database** to make changes in other databases. Thus, if both the *master* database and an other database reach their LCTs, to increase the size of the transaction

log in both databases, you would first need to use **alter database** to add log space to the *master* database, and then use it to add log space to the second database.

Changes to the *dump*, *load*, and *online* Commands

Because of the introduction of rollback records, version 11.9.2 makes the following changes in the way the **dump**, **load**, and **online** commands work:

- The **dump transaction** command has a new option, **with standby_access**, that only dumps completed transactions.
- The **online database** command has a new **for standby_access** option. You use the **for standby_access** option only after loading a transaction log that was originally dumped with the **with standby_access** option.
- The **load database** and **load transaction** commands initiate recovery operations, but do not complete them. To complete recovery, you must execute the **online database** command.

New *with standby_access* Option for the *dump transaction* Command

The **dump transaction** command has a new **with standby_access** option. Without the option, **dump transaction** dumps the entire transaction log, including records for all open transactions. When you use the **with standby_access** option, **dump transaction** dumps only completed transactions. It dumps the transaction log up to the furthest point at which there are no active transactions. This is illustrated in Figure 18-3.

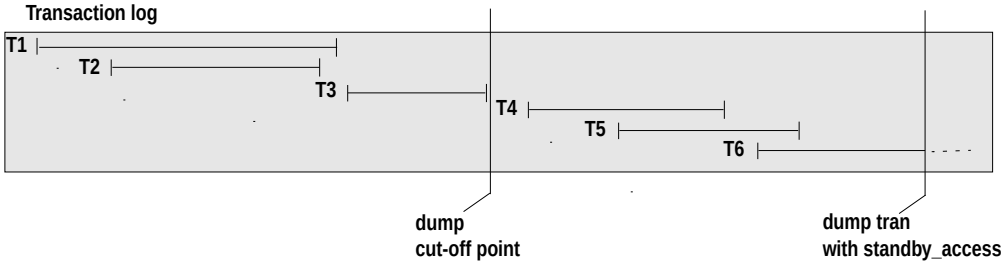

Figure 18-3: Dump cut-off point for dump transaction with standby_access

In Figure 18-3, a **dump transaction...with standby_access** command is issued at a point where transactions T1 through T5 have completed and transaction T6 is still open. Since the **with standby_access** clause causes a dump up to the furthest point at which there are no open transactions, the dump cannot go to the end of T5, because at that point, T6 is still open, and it cannot go to the end of T4, because at that point T5 is still open. Thus, the dump must stop at the end of transaction T3, where it will include completed transactions T1 through T3.

dump transaction Syntax

The new syntax for **dump transaction** is:

```
dump tran[saction] database_name to...
   [with standby_access]
```

You must use **dump tran[saction]...with standby_access** in all situations where you will be loading two or more transaction logs in sequence, and you want the database to be online between loads. This type of situation occurs when, for example, you have a read-only database that gets its data by loading transaction dumps from a primary database. In such a case, if the read-only database is used for generating a daily report based on transactions in the primary database, and the primary database's transaction log is dumped at the end of day, the daily cycle of operations is as follows:

1. On the primary database: **dump tran[saction]...with standby_access**
2. On the read-only database: **load tran[saction]...**
3. On the read-only database: **online database for standby_access**

◆ ***WARNING!***

If a transaction log contains open transactions, and you dump it without using the with standby_access option, version 11.9.2 will not allow you to load the log, bring the database online, and then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after loading a dump originally made with standby_access or after loading the entire series.

New *for standby_access* Option for the *online database* Command

The **online database** command has a new **for standby_access** option. You use **for standby_access** to bring a database online after loading it with a dump that was made using the **with standby_access** option.

◆ ***WARNING!***

If you try to use online database for standby_access with a transaction log that was not dumped using the with standby_access option, the command will fail.

Syntax

The new syntax for **online database** is:

```
online database database_name [for standby_access]
```

Changes to the *lct_admin* Function

The **lct_admin** function has been changed as follows:

- A new option, **abort**, replaces the **unsuspend** option of earlier versions.
- A new option, "**reserve", 0** displays the current last-chance threshold (LCT).

lct_admin abort

When the LCT is reached, all processes that might update the transaction log are suspended. In pre-11.9.2 versions, a user with System Administrator privileges can use **lct_admin** with the **unsuspend** option to reactivate suspended processes. This option is not available in version 11.9.2.

Version 11.9.2 replaces **lct_admin unsuspend** with **lct_admin abort**, which terminates transactions that have been suspended because of the LCT.

Syntax

The syntax for the new **abort** option is:

```
lct_admin("abort", process_id [, database_id])
```

You can use **lct_admin abort** to terminate the oldest open transaction or all open transactions in a transaction log that has reached it LCT. To terminate the oldest transaction, enter the process ID (*spid*) of the process that initiated the transaction (see "Using lct_admin abort" on page 18-9) to terminate all open transactions in the log, enter 0 for *process_id* and give the ID of the database that contains the log.

For example, if process 83 initiated the oldest open transaction in a suspended log, to terminate the transaction you would enter:

```
select lct_admin("abort", 83)
```

To terminate all open transactions in the log, you would enter:

```
select lct_admin("abort", 0, 12)
```

Using *lct_admin abort*

In pre-11.9.2 versions, a system administrator could use **lct_admin unsuspend** to reactivate transactions suspended because of the LCT. In version 11.9.2, **lct_admin unsuspend** is replaced by **lct_admin abort**, which terminates suspended transactions.

When the LCT is reached, all update processes are suspended until additional log space is made available. Typically, space is created by dumping the transaction log, since this removes closed transactions from the beginning of the log. However, if one or more transactions at the beginning of the log are still open, dumping the log will not help.

The purpose of **lct_admin abort** is to terminate suspended transactions at the beginning of a log. Since terminating a transaction closes it, this allows a new dump of the log to create free space by removing the transaction and possibly others after it. Figure 18-4 illustrates the type of situation in which you use **lct_admin abort**.

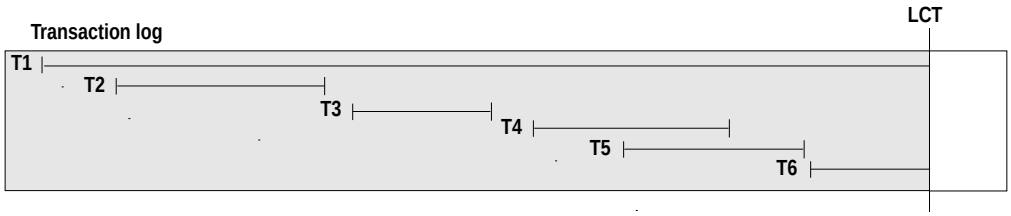

Figure 18-4: Example of when to use of lct_admin abort

In Figure 18-4, a transaction log has reached its LCT, and open transactions T1 and T6 are suspended. Because T1 is at the beginning of the log, it prevents a dump from removing closed transactions T3 through T5 and creating space for continued logging. Terminating T1 with **lct_admin abort** allows you to close T1 so that a dump can clear transactions T1 through T5 from the log.

To determine the ID of the oldest process in the transaction log, you can query the *syslogshold* table. See "The syslogshold Table" on page 21-32 of the *System Administration Guide* for more information.

lct_admin abort Syntax

The syntax for **lct_admin abort** is:

```
lct_admin("abort", {process_id [, database_id]})
```

You can use **lct_admin abort** to terminate the oldest open transaction or all open transactions in a transaction log that has reached its LCT.

To terminate the oldest transaction, enter the process ID (*spid*) of the process that initiated the transaction (see "Getting the Process ID for the Oldest Open Transaction" on page 18-10). Note that this also terminates any other suspended transactions in the log that belong to the specified process.

To terminate all open transactions in the log, enter:

```
select lct_admin("abort", 0, 12)
```

For example, if process 83 holds the oldest open transaction in a suspended log, and you want to terminate the transaction, enter:

```
select lct_admin("abort", 83)
```

This also terminates any other open transactions belonging to process 83 in the same transaction log.

To terminate all open transactions in the log, enter:

```
select lct_admin("abort", 0, 12)
```

Getting the Process ID for the Oldest Open Transaction

To get the process ID of the oldest open transaction in a log that has reached its LCT, use the *syslogshold* table in the *master* database, as shown in the following **isql** example:

```
1> use master
2> go
1> select dbid, spid from syslogshold
2> where dbid = db_id(name_of_database)
3> go
```

Using *lct_admin reserve* to Get the Current Last-Chance Threshold

When a database with shared log and data segments is initially created, its LCT is based on the size of the model database. As soon as data is added and logging activity begins, the LCT is recalculated dynamically, based on available space and currently open transactions. The LCT of a database with separate log and data segments is based on the size of the log segment and does not vary dynamically.

To get the current LCT of any database, you can use **lct_admin** with the **reserve** parameter and a specification of 0 log pages:

```
select lct_admin("reserve",0)
```

The LCT for a database is stored in the *systhresholds* table and is also accessible through **sp_helpthreshold**. However, the following points are to be noted:

- **sp_helpthreshold** returns user-defined thresholds and other data, as well as an up-to-date value for the LCT. Using **lct_admin** is simpler if you only need the current LCT.
- For a database with shared log and data segments, the LCT value in *systhresholds* may not be the current LCT value.

User-Defined Database Recovery Order

In version 11.9.2, the new system procedure **sp_dbrecovery_order** allows you to determine the order in which individual user databases recover. This makes it possible to assign a recovery order in which, for example, critical databases recover before lower-priority databases.

Previously, databases were recovered in a fixed order that the user could not alter. First system databases were recovered, then user databases. User databases were recovered in order, by database ID.

Important features of recovery order in version 11.9.2 are:

- System databases are recovered first, just as they were in earlier versions. The system databases and their order of recovery is:

- *master*
- *model*
- *tempdb*
- *sybsystemdb*
- *sybsecurity*
- *sybsystemprocs*

All other databases are considered user databases, and their recovery order can be specified by the user.

- You can use **sp_dbrecovery_order** to specify the recovery order of user-databases and to list the user-defined recovery order of an individual database or of all databases.
- User databases that are not explicitly assigned a recovery order with **sp_dbrecovery_order** are recovered according to their database ID, after all the databases that have a user-defined recovery order.
- If you do not use **sp_dbrecovery_order** to assign any databases a recovery order, user databases are recovered in order of database ID.

Using *sp_dbrecovery_order*

To use **sp_dbrecovery_order** to enter or modify a user-defined recovery order, you must be in the *master* database and have System Administrator privileges. Any user, in any database, can use **sp_dbrecovery_order** to list the user-defined recovery order of databases.

sp_dbrecovery_order Syntax

The syntax for **sp_dbrecovery_order** is:

```
sp_dbrecovery_order
   [database_name [, rec_order [, force]]]
```

where *database_name* is the name of the user database to which you want to assign a recovery order, and *rec_order* is the order in which the database is to be recovered.

Recovery order must be consecutive, starting with 1 and containing no gaps between values. The first database assigned a recovery order must be assigned a *rec_order* of 1. If you assign a recovery order to three databases, you must enter their recovery order as 1, 2, 3. You

cannot assign a recovery sequence of 1, 2, 4, with the intention of assigning a recovery order of 3 to another database at a later time.

To insert a database into a user-defined recovery sequence without putting the database at the end, you enter *rec_order* and specify **force**. For example, if databases *A*, *B*, and *C* have a user-defined recovery order of 1, 2, 3, and you want to insert the *pubs2* database as the second user database to recover, enter:

```
sp_dbrecovery_order pubs2, 2, force
```

This command assigns a recovery order of 3 to database *B* and a recovery order of 4 to database C.

Changing or Deleting the Recovery Position of a Database

To change the position of a database in a user-defined recovery sequence, you need to delete the database from the recovery sequence and then insert it in the position you want it to occupy. If the new position is not at the end of the recovery order, you need to use the **force** option.

To delete a database from a recovery sequence, specify a recovery order of -1.

For example, to move the *pubs2* database from recovery position 2 to recovery position 1, you delete the database from the recovery sequence and then reassign it a recovery order as follows:

```
sp_dbrecovery_order pubs2, -1
sp_dbrecovery_order pubs2, 1, "force"
```

Listing the User-Assigned Recovery Order of Databases

To list the recovery order of all databases assigned a recovery order through **sp_dbrecovery_order**, use **sp_dbrecovery_order** with no arguments:

```
sp_dbrecovery_order
```

This generates output such as the following:

```
The following databases have user specified
recovery order:
Recovery Order Database Name        Database Id
-------------- -------------------- -----------
    1          dbccdb                   8
    2          pubs2                    5
    3          pubs3                    6
    4          pubtune                  7
The rest of the databases will be recovered in
default database id order.
```

To display the recovery order of a specific database, enter **sp_dbrecovery_order** with the name of the database, as in the following **isql** example:

```
1> sp_dbrecovery_order pubs2
2> go
Database Name Database id Recovery Order
 ------------- ----------- --------------
 pubs2             5          2
```

Index-Level Fault Isolation for Data-Only-Locked Tables

When pages of an index for a data-only-locked table are marked as suspect during recovery, the entire index is taken offline. Two new system procedures manage offline indexes:

- **sp_listsuspect_object**
- **sp_forceonline_object**

In most cases, a System Administrator uses **sp_forceonline_object** to make a suspect index available only to those with the **sa_role**. If the index is on a user table, the suspect index can be repaired by dropping and re-creating the index.

See “sp_forceonline_object” on page 14-10 and “sp_listsuspect_object” on page 14-16 for more information.

For more information on recovery fault isolation, see the *System Administration Guide.*

Verifying Faults with *dbcc checkverify*

The fault verification operation **dbcc checkverify** is a companion command to **dbcc checkstorage**. The **dbcc checkverify** command examines

the results of the most recent **checkstorage** operation and reclassifies each soft fault as either a hard fault or an insignificant fault. **checkverify** acts as a second filter to remove spurious faults from the **checkstorage** results.

For information on using **dbcc checkstorage**, see Chapter 18, "Checking Database Consistency," in the *System Administration Guide.*

How *dbcc checkverify* Works

checkverify reads the recorded faults from the *dbcc_faults* table and resolves each soft fault through a procedure similar to that used by the **checkstorage** operation.

➤ ***Note***

Unlike **checkstorage**, the **checkverify** operation locks the table against concurrent updates. This ensures that the soft faults are reclassified correctly. **checkverify** does not find errors that have occurred since the last run of **checkstorage**.

checkverify records information in the *dbcc_operation_log* and *dbcc_operation_results* tables in the same way that **checkstorage** does. The recorded value of *opid* is the same as the *opid* of the last **checkstorage** operation. **checkverify** updates the *status* column in the *dbcc_faults* table and inserts a row in the *dbcc_fault_params* table for the faults it processes.

checkverify does not use the *scan* or *text* workspaces.

Each fault found by **checkstorage** is verified by **checkverify** as one of the following:

- A hard fault classified as such by **checkstorage**
- A soft fault reclassified as hard by **checkverify** because concurrent activity was ruled out as the cause
- A soft fault confirmed to be soft by **checkverify.** Some soft faults that appear when there is no concurrent activity in the database do not represent a significant hazard and are not reclassified as hard. A soft fault is not reclassified if it was informational only and not a corruption.
- A soft fault reclassified as insignificant because it can be attributed to concurrent activity or because subsequent activity masked the original inconsistency.

A fault that is assigned code 100011 (text pointer fault) by **checkstorage** is verified as hard if the text column has a hard fault. If it does not, it is reclassified as a soft fault.

A fault that is assigned code 100016 (page allocated but not linked) by **checkstorage** is verified as hard if the same fault appears in two successive **checkstorage** operations. Otherwise, it is reclassified as a soft fault.

When a fault that is assigned code 100035, spacebits mismatch, by **checkstorage** is verified as hard by **checkverify**, you can repair the fault by using **dbcc checktable**. For information on fixing these faults, see the section "Changes to dbcc checktable" on page 18-17.

When **checkverify** confirms hard faults in your database, follow the same procedures as you did in version 11.5 to correct the faults.

checkverify classifies the following fault codes as soft faults:

- 100020 – check aborted
- 100025 – row count fault
- 100028 – page allocation off current segment

When to Use *dbcc checkverify*

Hard faults are persistent, so they can be classified reliably by **checkverify**. You can verify persistent faults by running **checkverify** anytime after running **checkstorage**, even after an extended period of hours or days. However, when deciding your schedule, keep in mind that the database state changes over time, and the changes can mask both soft faults and hard faults.

For example, having a page that is linked to a table but not allocated is a hard fault. If the table is dropped, the fault is resolved and masked. If the page is allocated to another table, the fault persists but its signature changes. The page now appears to be linked to two different tables. If the page is reallocated to the same table, the fault appears as a corrupt page chain.

Persistent faults that are corrected by a subsequent database change usually do not pose an operational problem. However, detecting and quickly verifying these faults may help locate a source of corruption before more serious problems are encountered or before the signature of the original fault changes. For this reason, Sybase recommends that you run **checkverify** as soon as possible after running **dbcc checkstorage**.

➤ ***Note***

When **checkstorage** is executed with the target database in single-user mode, there will be no soft faults and no need to execute **checkverify**.

checkverify runs only one time for each execution of **checkstorage**. However, if **checkverify** is interrupted and does not complete, you can run it again. The operation resumes from where it was interrupted.

How to Use *dbcc checkverify*

The syntax is:

dbcc checkverify(*dbname*)

where *dbname* is the name of the database for which you want to verify **checkstorage** results.

checkverify operates on the results of the last completed **checkstorage** operation for the specified database only.

When the **checkverify** operation is complete, Adaptive Server returns the following message:

```
DBCC checkverify for database name, sequence
n completed at date time. n suspect conditions
resolved as faults and n resolved as innocuous.
n checks were aborted.
```

You can run **checkverify** automatically after running **checkstorage** by using **sp_dbcc_runcheck**.

Changes to *dbcc checktable*

When **checkstorage** returns a fault code of 100035, and **checkverify** confirms that the spacebit fault is a hard fault, you can use the new option for **dbcc checktable** to fix the reported fault.

The syntax is:

dbcc checktable (*table_name*, fix_spacebits)

where *table_name* is the name of the table to repair.

Estimating *number of locks* for Data-Only-Locked Tables

Changing to data-only locking may require more locks or may reduce the number of locks required:

- Tables using datapages locking require fewer locks than tables using allpages locking, since queries on datapages-locked tables do not acquire separate locks on index pages.
- Tables using datarows locking can require a large number of locks. Although no locks are acquired on index pages for datarows-locked tables, data modification commands that affect many rows may hold more locks. Queries running at transaction isolation level 2 or 3 can acquire and hold very large numbers of row locks.

Insert Commands and Locks

An insert with allpages locking requires *N*+1 locks, where *N* is the number of indexes, plus one lock for the data page. The same insert on a data-only-locked table locks only the data page or data row.

select Queries and Locks

Scans at transaction isolation level 1, with **read committed with lock** set to hold locks, acquire overlapping locks that roll through the rows or pages, so they hold, at most, two data page locks at a time. However, transaction isolation level 2 and 3 scans, especially those using datarows locking, can acquire and hold very large numbers of locks, especially when running in parallel. Using datarows locking, and assuming no blocking of an attempt at lock promotion, the maximum number of locks that might be required for a single table scan is:

```
row lock promotion HWM * parallel_degree
```

If lock contention from exclusive locks prevents scans from promoting to a table lock, the scans can acquire a very large number of locks. Instead of configuring the number of locks to meet the extremely high locking demands for these queries, consider changing applications that affect large numbers of rows at transaction isolation level 2 or 3 to use the **lock table** command. This command acquires a table lock without attempting to acquire individual page locks. See “Explicitly Locking a Table with the lock table Command” on page 7-1 for information on using **lock table**.

Data Modification Commands and Locks

For tables that use the datarows locking scheme, data modification commands can require many more locks. For example, a transaction that performs a large number of inserts into a heap table may acquire only a few page locks for an allpages-locked table, but will require one lock for each inserted row in a datarows-locked table. Similarly, transactions that update or delete large numbers of rows may acquire many more locks with datarows locking.

Using the License Use Monitor

The License Use Monitor allows a System Administrator to monitor the number of user licenses used in Adaptive Server. By monitoring the number of user licenses, you can securely manage the license agreement data. That is, you can ensure that the number of licenses used on your Adaptive Server does not exceed the number specified in your license agreement.

The License Use Monitor only monitors the number of licenses issued; it does not enforce the license agreement. If the License Use Monitor reports that you are using more user licenses than specified in your license agreement, see your Sybase sales representative.

You must have System Administrator privileges to configure the License Use Monitor.

By default, the License Use Monitor is turned off when Adaptive Server is first installed or upgraded. The System Administrator must configure the License Use Monitor before it can monitor license usage. See ““Configuring License Manager to Monitor User Licenses” on page 18-21 for configuration information.

How Licenses Are Counted

A license consists of the combination of a host computer name and a user name. If a user logs in to Adaptive Server multiple times from the same host machine, it counts as one license. However, if the user logs in once from host A, and once from host B, it counts as two licenses. If multiple users log in to Adaptive Server from the same host, but with different user names, each distinct combination of user name and host name is counted.

Monitoring License Use with the Housekeeper Task

After you configure the License Use Monitor, the housekeeper task determines how many user licenses are in use, based on the user ID and the host name of each user logged in to Adaptive Server. When the housekeeper task checks licenses, the License Use Monitor updates a variable that tracks the maximum number of user licenses in use:

- If the number of licenses in use is the same or has decreased since the previous housekeeper run, the License Use Monitor does nothing
- If the number of licenses in use has increased since the previous housekeeper run, the License Use Monitor sets this number as the maximum number of licenses in use
- If the number of licenses in use is greater than the number allowed by the license agreement, the License Use Monitor issues the following error message to the error log:

  ```
  Exceeded license usage limit. Contact Sybase Sales
  for additional licenses.
  ```

The housekeeper task runs during Adaptive Server's idle cycles. The housekeeper only monitors the number of user licenses if the **housekeeper free write percent** configuration parameter is set to 1 or greater.

For more information about the housekeeper task, see Chapter 21, "How Adaptive Server Uses Engines and CPUs," in the *Performance and Tuning Guide* and Chapter 18, "System Administration Changes" in this book.

Logging the Number of User Licenses

The *syblicenseslog* system table is created in the *master* database during installation or upgrade of Adaptive Server. The License

Manager updates the columns in *syblicenseslog* at the end of each 24-hour period, as shown in Table 18-1.

Table 18-1: Columns in *syblicenseslog* table

Column	Description
status	One of the following: -1 – housekeeper unable to monitor licenses 0 – number of licenses not exceeded 1 – number of licensees exceeded
logtime	Date and time the log information was inserted.
maxlicenses	Maximum number of licenses used during the previous 24 hours.

syblicenseslog looks similar to the following:

```
status logdate                    maxlicenses
 ------ -------------------------- -----------
      0        Jul 17 1998 11:43AM         123
      0        Jul 18 1998 11:47AM         147
      1        Jul 19 1998 11:51AM         154
      0        Jul 20 1998 11:55AM         142
      0        Jul 21 1998 11:58AM         138
      0        Jul 21 1998  3:14PM         133
```

In this example, the number of user licenses used exceeded the limit on July 19, 1998. The second row for July 21, 1998 was caused by a shutdown and reboot of the server.

If Adaptive Server is shut down, License Manager updates *syblicenseslog* with the current maximum number of licenses used. Adaptive Server starts a new 24-hour monitoring period when it is rebooted.

Configuring License Manager to Monitor User Licenses

Use **sp_configure** to specify the number of licenses in your license agreement:

```
sp_configure "license information" , number
```

where *number* is the number of licenses. For example:

```
sp_configure "licenses information", 300
```

sets the maximum number of user licenses to 300, and reports an overuse of licenses for license number 301. If you increase the

number of user licenses, you must also change the **license number** configuration parameter.

The configuration parameter **housekeeper free write percent** must be set to 1 or more in order for the License Manager to track license use.

New Housekeeper Tasks

In pre-11.9.2 Adaptive Server, the housekeeper task checks caches for changed pages and writes those pages to disk. The number of writes that the housekeeper is allowed to initiate is controlled by the configuration parameter **housekeeper free write percent.**

In version 11.9.2, the housekeeper performs these new tasks:

- Reclaims space in data-only-locked tables and indexes, by clearing space used by committed deletes
- Flushes statistics generated from data modification operations from in-cache memory to *systabstats*
- Monitors license use, if **license information** is set to a value greater than 0

The tasks are performed in round-robin fashion; each time the housekeeper wakes up, it performs the next task in the cycle. The new tasks generate log records and acquire locks.

Disabling the Housekeeper Task

If the value of **housekeeper free write percent** is set to 0, the housekeeper does not run and does not perform any tasks. You should set this value to 0 only if disk contention on your system is high, and it cannot tolerate the extra I/O generated by the housekeeper.

◆ ***WARNING!***

Setting housekeeper free write percent to 0 disables flushing statistics to the *systabstats* table. This can seriously impair performance if statistics change significantly.

If you disable the housekeeper tasks by setting **housekeeper free write percent** to 0, you need to be certain that statistics are kept current. Commands that write statistics to disk are:

- **update statistics**

- **dbcc checkdb** (for all tables in a database) or **dbcc checktable** (for a single table)
- **sp_flushstats**

You should run one of these commands on any tables that have been updated since the last time statistics were written to disk, at the following times:

- Before dumping a database
- Before an orderly shutdown
- After rebooting, following a failure or orderly shutdown; in these cases, you cannot use **sp_flushstats**, you must use **update statistics** or **dbcc** commands
- After any significant changes to a table, such as a large bulk copy operation, altering the locking scheme, deleting or inserting large numbers of rows, or a **truncate table** command.

Reclaiming Space

When a user deletes rows from data-only-locked tables, a task is queued to the housekeeper to check that page for space reclamation.

The configuration parameter **enable housekeeper GC** must be set to 1, its default value, in order for the housekeeper to perform this task. Also, the configuration parameter **housekeeper free write percent** must be set to 1 or greater.

If **enable housekeeper GC** is set to 0, the housekeeper no longer performs space reclamation. If all tables on your server use the allpages locking scheme, or if very few deletes or shrinking updates are performed on data-only-locked tables, setting **enable housekeeper GC** to **0** improves performance by slightly reducing housekeeper overhead.

Flushing Statistics

The table and index statistics that are used to optimize queries are maintained in memory structures during query processing. When these statistics change, the changes are not written to the *systabstats* table immediately, to reduce I/O contention and improve performance. Instead, the housekeeper task periodically flushes statistics to disk. The configuration parameter **housekeeper free write percent** must be set to 1 or greater.

Setting **housekeeper free write percent** to 0 disables the automatic flushing of statistics to disk. Use this setting only if your system cannot tolerate the I/O contention caused by the additional housekeeper writes.

◆ ***WARNING!***

Disabling the flushing of statistics can seriously impair query performance.

Information on Housekeeper Tasks

The system procedure **sp_sysmon** reports on how often housekeeper tasks run and the work performed by the tasks. See "Housekeeper Task Activity" on page 20-6.

European Currency Symbol

Adaptive Server supports the use of the European currency symbol, or "Euro." The Euro character looks like this:

To support the use of this character, Adaptive Server identifies the Euro character in the character set support files.

Adaptive Server includes all the changes that were made by Microsoft to their Windows character sets. These changes are part of Windows NT 4 Server Pack 4, Windows 98, and Windows NT 5. To enable Adaptive Server to process the Euro character, you must be using Adaptive Server 11.9.2.

New Character Set – ISO 8859-15

ISO 8859-15 is a new character set. It is the same as iso_1 except for changes to the following code points:

Code Point	Changed To
0xA4	The Euro character
0xA6	Uppercase letter “S” with caron
0xA8	Lowercase letter “s” with caron
0xB4	Uppercase letter “Z” with caron
0xB8	Lowercase letter “z” with caron
0xBC	Uppercase OE ligature
0xBD	Lowercase OE ligature
0xBE	Uppercase Y diaeresis

Character Set Changes

The following changes were made to the character set files to support the Euro character:

- In code page 1250, code point 0x80 was changed to the Euro character. This change matches the change that Microsoft made to code page 1250. Code point 0x80 was an undefined character in previous versions of Adaptive Server.

  If you are using the undefined character previously assigned to 0x80 in *char* or *varchar* columns, you need to update your data to avoid a conflict.

- In code page 1251, code point 0x88 was changed to the Euro character. This change matches the change that Microsoft made to code page 1251. Code point 0x88 was an undefined character in previous versions of Adaptive Server.

  If you are using the undefined character previously assigned to 0x88 in *char* or *varchar* columns, you need to update your data to avoid a conflict.

- Changes to code page 1252 match the change to code page 1250. In addition, two new characters that Microsoft added to code page 1252 were added to Adaptive Server code page 1252. The new characters are:

  - Latin uppercase letter “Z” with caron (at code point 0x8E)
  - Latin lowercase letter “z” with caron (at code point 0x9E)
- Code pages 874, 1253, 1254, 1255, 1256, 1257, and 1258 were changed in the same way as code page 1250.

Adaptive Server Conversion Tables

The Unilib™ conversion tables for the affected character sets (cp1250 through cp1258, cp874, and ISO 8859-15) were updated to convert the new character.

Open Client Changes

Open Client™ conversion tables were updated for character sets cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, and cp874.

A new conversion table for ISO 8859-15 is available.

Character-Set Conversions That Change Data Lengths

In some cases, the character set used on a server is different from the character set used on a client, and converting data from the server’s character set to the client’s character set changes the length of the data. This is always the case, for example, when the character set on one system uses 1 byte to represent each character and the character set on the other system requires 2 bytes per character. In pre-11.9.2 versions, such conversions had to be carried out on the client. In version 11.9.2, the server can perform such conversions for all supported character sets.

When a server-to-client character-set conversion causes a change in data length, there are two possibilities:

- Server-to-client data length decreases

  Examples in which data length decreases from server to client are:

  - Multibyte UTF-8 Greek or Russian to a single-byte Greek or Russian character set
  - Japanese character-set conversion from 2-byte Hankaku-Katakana EUC-JIS to single-byte Shift-JIS.

- Server-to-client data length increases

  Examples in which data length increases from server to client are:

  - Single-byte Thai to multibyte UTF-8 Thai
  - Japanese character-set conversion from single-byte Shift-JIS to 2-byte Hankaku-Katakana EUC-JIS

In version 11.9.2:

- The server automatically and transparently carries out character-set conversions that decrease server-to-client data length.
- You can configure the server to carry out character-set conversions that increase server-to-client data length.

➤ ***Note***

In version 11.9.2, the **bcp**, **isql**, **defncopy** and **optdiag** utility programs can receive character-set conversions that reduce data length, but they cannot receive character-set conversions from the server that increase data length.

The remainder of this section describes server-performed character-set conversions that increase data length. For background information on supported character sets and character-set conversion, see *Configuring Adaptive Server for UNIX Platforms* and Chapters 13 and 14 in the *System Administration Guide.*

Conversions When Server-to-Client Data Length Increases

To enable and make use of character-set conversions on the server that increase data length:

- On the server, use the **sp_configure** command to enable Unicode conversions.
- On the client, you need to be able to handle CS_LONGCHAR data, and you need to inform the server of this capability at connection time, using the Open Client **ct_capability** function.

Configuring the Server

Adaptive Server version 11.5.1 introduced several enhancements related to character sets and character-set conversions. One of these enhancements was Unicode conversion, in which the server

performs character-set conversions by first converting its local character set to Unicode and then converting the resulting Unicode to the destination character set. Unicode conversion is necessary when the server is not able to perform a character-set conversion directly from one character set to another. It is available as a configuration option set through the **sp_configure** system procedure.

In version 11.5.1, Unicode conversion can be used only for character-set conversions that do not change data length. Version 11.9.2 allows Unicode conversions that change data length. In the case of server-to-client character-set conversions that decrease data length, the server automatically performs the conversion. In the case of server-to-client character-set conversions that increase data length, you need to configure the server explicitly by setting the **enable unicode conversions** parameter to either 1 or 2.

- If you set **enable unicode conversions** to 1:

  ```
  sp_configure "enable unicode conversions", 1
  ```

  Adaptive Server first follows pre-11.5.1 behavior and looks for a converter that directly maps the server's character set to the client's character set. Direct converters are not available for mappings that change data length. If it cannot find a direct converter, Adaptive Server carries out a Unicode conversion.

- If you set **enable unicode conversions** to 2:

  ```
  sp_configure "enable unicode conversions", 2
  ```

  Adaptive Server carries out a Unicode conversion, without attempting to find a converter for a direct character-set mapping.

Client Requirements

To make use of a server-performed character-set conversion that increases data length:

- A client application must use Client-Library version 11.1 or later.
- The client must be able to handle CS_LONGCHAR data.

  When the server carries out a character-set conversion that increases data length, *char* and *varchar* data is converted to the client's character set and sent to the client as CS_LONGCHAR data. Thus, to make use of server-side character-set conversions that increase data length, a client application must be coded to extract its own character set from data received as CS_LONGCHAR.

- In establishing a connection with a server, a client application must call the Open Client **ct_capability** function with the *capability* parameter set to CS_DATA_LCHAR and the *value* parameter set to CS_TRUE:

  ```
  CS_INT capval = CS_TRUE
  ct_capability(connection,CS_SET,CS_CAP_RESPONSE,
     CS_DATA_LCHAR,&capval)
  ```

  where *connection* is a pointer to a CS_CONNECTION structure.

Changes to Component Integration Services

19 What's New in Component Integration Services

This chapter describes new features and changes to existing functionality for Component Integration Services (CIS) for Adaptive Server version 11.9.2. Topics include:

The new features of CIS for Adaptive Server 11.9.2 enhance the performance of queries that include remote tables, and simplify the mapping of local proxy tables to remote tables.

Performance Enhancements

Adaptive Server 11.9.2 introduces the following performance enhancements for queries that include remote tables:

- Improvements to the query optimizer
- Changes to the quickpass optimization strategy

Improvements to the Query Optimizer

Improvements have been made to the query optimizer which result in faster queries on remote tables. By evaluating the cost of network access, the optimizer now favors the query plans described in the following scenarios:

- When a single remote table is being joined to one or more local tables, the remote table is typically positioned as the outermost table in the query plan, and the selected rows are streamed efficiently across the network in one pass.
- When several remote tables are located in the same remote database and are joined with non-remote tables, the remote tables are typically positioned adjacent to each other as the outermost tables in the plan. At execution time, the remote tables can often

be joined at the remote database, and only the result of that subjoin is streamed across the network in one pass.

- When remote tables from more than one database are included in a query (possibly with local tables as well), the tables from one remote database are typically positioned as the outermost tables in the plan. The other remote tables are accessed via the reformatting approach.

Reformatting Approach

The reformatting approach has two phases:

- Rows are selected from the remote table, streamed across the network in one pass and loaded into a temporary worktable with a clustered index on the join key. This is performed for each remote table that is to be reformatted prior to the execution of the main query plan.
- Then, as part of the main query plan, these worktables are joined in the same fashion as a local table with an index on the join key.

This reformatting approach is of great value when you are accessing remote tables that are positioned as inner tables in the query plan, since it allows the selected rows to be streamed across the network in one pass. This approach has many of the benefits of merge join.

Changes to Quickpass Optimization Strategy

In pre-11.9.2 versions, the quickpass optimization strategy examines the query tree generated by a SQL command and determines whether the entire command can be forwarded to a single remote server. If the entire command can be forwarded, CIS reconstructs the SQL statement from the query trees into the syntax required by the remote server, and produces a query plan that contains the text of that statement, plus instructions for routing its results. If the entire command cannot be forwarded, none of the syntax is forwarded to the remote server.

CIS for Adaptive Server 11.9.2 includes the following enhancements to the quickpass strategy:

- Removal of some 11.5 quickpass restrictions
- Additional analysis of the statement to determine if portions of the statement can be sent to the remote server, if the entire statement cannot be forwarded to a remote server

Removal of 11.5 Quickpass Restrictions

In pre-11.9.2 versions, CIS automatically excludes the following type of statements from quickpass mode:

- Statements that include *text* or *image* columns
- Statements that include a **declare cursor** command

In version 11.9.2, CIS evaluates these statements to determine if they can be handled in quickpass mode.

Additional Analysis of the Statement

In Adaptive Server 11.9.2, if the entire statement cannot be forwarded to the remote server, CIS analyzes clauses within that statement to determine whether the syntax represented by the clause can be forwarded to the remote server. The statements that significantly benefit from the additional analysis are:

- Statements involving the **union** operator, where the **select** statement on one side of the union references tables that reside on a different server than the **select** statement on the other side of the union. Quickpass mode analyzes each side of the union, and sends each individual **select** statement to a remote server if possible. For example, the following query is a union between Oracle and DB2 tables:

  ```
  select name, sum(salary) as salary
  from oracle_t1, oracle_t2
  where oracle_t1.id = oracle_t2.id
  group by name
  union
  select name, sum(salary) as salary
  from db2_t1, db2_t2
  where db2_t1.id = db2_t2.id
  group by name
  ```

  Quickpass mode sends the first **select** statement to the Oracle server and the second **select** statement to the DB2 server.

- Statements involving a **select...into** command. **select...into** is a compound statement involving two steps:

  - Create the target table
  - Execute an **insert...select** operation

  Quickpass mode analyzes the **insert...select** portion of the statement separately and handles it in quickpass mode if possible. For example, in pre-11.9.2 versions, the following

grouped aggregate operation cannot be forwarded to a remote server:

```
select name, sum(salary) as salary
into #summary
from t1, t2
where t1.id = t2.id
group by c1
```

Assuming that both *t1* and *t2* reside on the same remote server, quickpass mode converts this statement into the following two statements, and sends the **insert...select** statement to the remote server:

```
create table #summary
(name varchar(30), salary money)
```

```
insert into #summary (name, salary)
select name, sum(salary) as salary
from t1, t2
where t1.id = t2.id
group by name
```

Restrictions on Sending Partial Queries to a Remote Server

Quickpass mode does not send partial queries to a remote server when:

- The tables in the partial query reference two or more separate servers
- The partial query contains a subquery
- The partial query contains *text* or *image* columns
- The original query references a view containing grouped aggregates, or the view participates in grouped aggregate operations

System Changes in Component Integration Services

The 11.9.2 version of CIS includes the following system changes:

- New server classes
- New method for mapping remote objects to local proxy tables
- Updatable vs. read-only tables

New Server Classes

Three new server classes have been added for configuring a server with **sp_addserver**. In pre-11.9.2 versions of Adaptive Server, all Sybase servers were configured as server class *sql_server*. The new server classes allow CIS to forward more syntax to the remote Sybase server. The new classes are:

- *ASEnterprise* – Adaptive Server Enterprise version 11.5 or later. This replaces *sql_server* as the default server class.
- *ASAnywhere* – Adaptive Server Anywhere™ version 6.0 or later.
- *ASIQ* – Adaptive Server IQ™ version 11.5.1 or later.

The server class *sql_server* is still used for Sybase SQL Server™.

In pre-11.9.2 versions of Adaptive Server, server class *sds* is an alias for server class *direct_connect*. In version 11.9.2, server class *sds* is a class of its own.

Updating Existing Server Classes

If you have existing servers that are class *ASEnterprise*, *ASAnywhere*, or *ASIQ*, Sybase recommends that you change the server's class in the *sysservers* table from *sql_server* to the correct server class.

Re-run the **sp_addserver** system procedure to change a server's class. For example, to change an Adaptive Server named MYSVR from server class *sql_server* to server class *ASEnterprise*, enter:

```
sp_addserver MYSVR, ASEnterprise
```

➤ ***Note***

If you have a Specialty Data Store server that is defined as server class *direct_connect*, make sure you change its class to *sds*.

Configuring Databases In Adaptive Server Anywhere

If an Adaptive Server Anywhere server is configured to support multiple databases, you must define each database to Adaptive Server Enterprise as if it were a separate remote server. Adaptive Server Anywhere derives the database that should be used from the server name that a client is using for the connection.

If an Adaptive Server Anywhere server supports only a single database, that database will always be the default.

For example, if an instance of Adaptive Server Anywhere supports three separate databases named DB1, DB2, and DB3, the interfaces file should look something like this:

```
DB1
    query tcp ether hostname 2638
DB2
    query tcp ether hostname 2638
DB3
    query tcp ether hostname 2638
```

The host name and port number are identical, but the server name used by clients to connect to Adaptive Server Anywhere is either DB1, DB2, or DB3. Within Adaptive Server Enterprise, define each server:

```
exec sp_addserver DB1, ASAnywhere, DB1
exec sp_addserver DB2, ASAnywhere, DB2
exec sp_addserver DB3, ASAnywhere, DB3
```

When using server class *ASAnywhere*, configure Adaptive Server Anywhere to emulate Adaptive Server Enterprise to ensure compatibility. This can be done in two ways:

- Use Sybase Central™ to create a database in Adaptive Server Anywhere, and select the Emulate Adaptive Server option
- Use **dbinit** to create a database in Adaptive Server Anywhere, and choose the “Ignore Trailing Blanks” and “Case Sensitivity for Names and Values” flags

For more information, see the *Adaptive Server Anywhere User’s Guide*.

New Method for Mapping Remote Objects to Local Proxy Tables

In pre-11.9.2 versions of Adaptive Server, you use the **sp_addobjectdef** or **sp_defaultloc** system procedure to map a remote object to a local proxy table. With version 11.9.2, you can map the remote object to a proxy table when you create that proxy table, using one of these commands:

- **create table** – creates the table in the remote location and then creates the proxy table. Use **create table** if the table does not exist at the remote location. For more information, see “create table” on page 12-22.

- **create existing table** – creates the proxy table using the column list you specify. Use **create existing table** only if the table exists at the remote location. For more information, see "Changes to create existing table" on page 19-8.
- **create proxy_ table** – creates the proxy table using a column list that CIS derives, based on metadata it obtains from the remote location. Use **create proxy_table** only if the table exists at the remote location. For more information, see "create proxy_table" on page 12-19.

Adaptive Server 11.9.2 still supports the **sp_addobjectdef** and **sp_defaultloc** system procedures.

Updatable vs. Read-Only Cursors

By default, when a **declare cursor** command does not specify **for update** or **for read only,** Adaptive Server assumes that the cursor is updatable. This is consistent with ANSI specifications, but has significant performance implications when any table in the command is external to the local Adaptive Server. A new trace flag in version 11.9.2 overrides this default. If trace flag 11218 is turned on, any query that is part of a **declare cursor** command, and that references proxy tables, is read only by default. If an updatable cursor is desired when trace flag 11218 is on, explicitly provide the **for update** clause in the **declare cursor** command.

The trace flag can be set for a session with:

```
dbcc traceon(11218)
```

To set the trace flag server-wide, include it in the runserver file.

On UNIX:

```
dataserver -dmaster_dev_name -T11218
```

On NT:

```
sqlserver.exe -dmaster_dev_name -T11218
```

Transact-SQL Changes

CIS for Adaptive Server 11.9.2 introduces changes to the following Transact-SQL commands:

- **create existing table**

- **create proxy_table**
- **create table**

Changes to *create existing table*

create existing table has the following changes in Adaptive Server 11.9.2:

- Allows the use of the keywords **at**, **external**, **table**, and **procedure** to map the proxy table to a table, view, or procedure at a remote location
- Modifications to how server classes handle mapping proxy table columns to remote table columns
- Allows input parameter columns in remote procedure definitions

For more information, see "create existing table" on page 12-10.

New *create proxy_table* Command

create proxy_table creates a proxy table without specifying a column list. CIS derives the column list from the metadata it obtains from the remote table. For more information, see "create proxy_table" on page 12-19.

Changes to *create table*

create table allows the use of the keywords **at**, **external**, and **table** to map a proxy table to a table or view at a remote location. For more information, see "create table" on page 12-22.

Sybase Central Replaces the *ddlgen* and *defgen* Utilities

The **ddlgen** and **defgen** utilities do not exist in Adaptive Server 11.9.2. Use Sybase Central to back up data definition statements associated with remote servers, and to automatically create proxy tables for remote objects.

New Error Messages

CIS for Adaptive Server 11.9.2 includes the following new error messages:

Message 11270, severity 16, state 1:

```
Column '%.*s' does not allow NULL. Any column
defined as a parameter column for RPC tables must
allow NULL. A parameter column is a column whose
name begins with an underscore.
```

This message appears when you create a type procedure table and a parameter column definition does not specify NULL.

Message 11271, severity 16, state 1:

```
Column '%.*s' must precede all parameter columns.
A parameter column is a column whose name begins
with an underscore.
```

This message appears when you create a type procedure table and parameter columns (column names that begin with an underscore character, _) precede regular result columns—all column names that begin with an underscore must appear at the end of the list.

Message 11272, severity 16, state 1:

```
Action requested is not valid for remote tables
(%.*s).
```

This message appears when you enter the following commands using proxy tables (the commands can only be used with local tables):

- **alter table** ***table_name*** **lock** ***lock_scheme***
- **alter table** ***table_name*** **partition** ***n***

Message 11273, severity 16, state 1:

```
Encounter %d conversion errors during processing
of external statistics; some rows have been
ignored.
```

This message appears when **update statistics cannot** to generate complete statistics because the proxy table has columns with datatypes or widths that are different from the corresponding column at the remote location.

Component Integration Services Configuration

CIS is an integral part of Adaptive Server 11.9.2. Therefore, you do not need Extended Enterprise Option to access remote, heterogeneous database systems. The **enable cis** configuration parameter determines whether CIS functions are accessible to Adaptive Server.

Performance and Tuning Issues

20 Enhancements to *sp_sysmon*

This chapter contains the following sections:

Lock Management

Additions to the "Lock Management" section of the **sp_sysmon** report are as follows:

- The number of times the lock hash table was checked for locks and the average length of the hash chains are reported in the summary information.
- Row locks and deadlocks on datarows-locked tables are reported in the "Lock Detail" section.
- Row lock to table lock promotions are reported.
- The number of times that lock requests exceeded a lock timeout period are reported in "Timeouts by Lock Type".

Lock Hash Table Information

The "Lock Summary" report at the top of this section now reports:

- The number of times the lock hash table was searched for a lock on a page, row, or table
- The average length of the hash chains in the lock hash table

Sample Output for Lock Hash Table Information

```
Lock Management
---------------

  Lock Summary            per sec      per xact       count  % of total
  -----------------  ------------  ------------  ----------  ----------
  Total Lock Requests      2634.5         151.2     1580714       n/a
  Avg Lock Contention         2.4           0.1        1436       0.1 %
  Deadlock Percentage         0.0           0.0           1       0.0 %
  Lock Hashtable Lookups   8262.3         474.2     4957363       n/a
  Avg Hash Chain Length       n/a           n/a     0.01153       n/a
```

The "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 4, consider increasing the size of the hash table. See "lock hashtable size" on page 16-3 for more information.

Row Lock Information

The "Lock Detail" section has three blocks of information reporting on exclusive, update, and shared row locks. This output displays 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                per sec       per xact  count     % of total
-------------------------  ------------  --------  --------  ----------
Exclusive Row
    Granted                       1.3         0.1       751      75.6 %
    Waited                        0.4         0.0       243      24.4 %
  -----------------------  ---------  ---------  -------  ----------
  Total EX-Row Requests           1.7         0.1       994       0.1 %

  Update Row
    Granted                       0.2         0.0        95      38.0 %
    Waited                        0.3         0.0       155      62.0 %
  -----------------------  ---------  ---------  -------  ----------
  Total UP-Row Requests           0.4         0.0       250       0.0 %

  Shared Row
    Granted                    1699.8       103.3   1019882     100.0 %
    Waited                        0.1         0.0        46       0.0 %
  -----------------------  ---------  ---------  -------  ----------
  Total SH-Row Requests        1699.9       103.3   1019928      59.7 %
```

Lock Timeout Information

The "Lock Timeouts 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 timeout. The detail rows that show the lock types are printed only if lock timeouts occurred during the sample period. If no lock timeouts occurred, the "Total Lock Timeouts" row is displayed with all values equal to 0.

Sample Output for Lock Timeouts

```
Lock Timeouts by Lock Type   per sec   per xact    count  % of total
-------------------------  ---------  ---------  -------  ----------
  Exclusive Table                0.0        0.0        0       0.0 %
  Shared Table                   0.0        0.0        0       0.0 %
  Exclusive Intent               0.0        0.0        4      44.4 %
  Shared Intent                  0.0        0.0        0       0.0 %
  Exclusive Page                 0.0        0.0        0       0.0 %
  Update Page                    0.0        0.0        1      11.1 %
  Shared Page                    0.0        0.0        4      44.4 %
  Exclusive Row                  0.0        0.0        0       0.0 %
  Update Row                     0.0        0.0        0       0.0 %
  Shared Row                     0.0        0.0        0       0.0 %
  Exclusive Address              0.0        0.0        0       0.0 %
  Shared Address                 0.0        0.0        0       0.0 %
  Shared Next-Key                0.0        0.0        0       0.0 %
-------------------------  ---------  ---------  -------
Total Lock Timeouts              0.0        0.0        9
```

For more information on lock timeouts, see "Session-Level and System-Level Time Limits on Waiting for a Lock" on page 7-3.

Transaction Profile

The "Transaction Profile" section reports on data modifications by type of command and table locking scheme.

Sample Output for Transaction Profile

```
Transaction Profile
-------------------

  Transaction Summary          per sec   per xact    count  % of total
  -------------------------  ---------  ---------  -------  ----------
    Committed Xacts               16.5        n/a     9871     n/a

  Transaction Detail           per sec   per xact    count  % of total
  -------------------------  ---------  ---------  -------  ----------
    Inserts
      APL Heap Table             229.8       14.0   137900      98.6 %
      APL Clustered Table          2.5        0.2     1511       1.1 %
      Data Only Lock Table         0.9        0.1      512       0.4 %
  -------------------------  ---------  ---------  -------  ----------
    Total Rows Inserted          233.2       14.2   139923      91.5 %

    Updates
      APL Deferred                 0.5        0.0      287       2.3 %
      APL Direct In-place          0.0        0.0       15       0.1 %
      APL Direct Cheap             0.0        0.0        3       0.0 %
      APL Direct Expensive         0.0        0.0        0       0.0 %
      DOL Deferred                 0.4        0.0      255       2.1 %
      DOL Direct                  19.7        1.2    11802      95.5 %
  -------------------------  ---------  ---------  -------  ----------
    Total Rows Updated            20.6        1.3    12362       8.1 %

    Data Only Locked Updates
      DOL Replace                 19.6        1.2    11761      97.6 %
      DOL Shrink                   0.0        0.0        1       0.0 %
      DOL Cheap Expand             0.3        0.0      175       1.5 %
      DOL Expensive Expand         0.2        0.0      101       0.8 %
      DOL Expand & Forward         0.0        0.0       18       0.1 %
      DOL Fwd Row Returned         0.0        0.0        0       0.0 %
  -------------------------  ---------  ---------  -------  ----------
    Total DOL Rows Updated        20.1        1.2    12056       7.9 %

    Deletes
      APL Deferred                 0.5        0.0      308      48.4 %
      APL Direct                   0.0        0.0        9       1.4 %
      DOL                          0.5        0.0      320      50.2 %
  -------------------------  ---------  ---------  -------  ----------
    Total Rows Deleted             1.1        0.1      637       0.4 %
  =========================  =========  =========  =======
    Total Rows Affected          254.9       15.5   152922
```

Inserts

"Inserts" displays the number of inserts performed on:

- Allpages-locked heap tables
- Allpages-locked tables with clustered indexes
- Data-only locked tables

➤ ***Note***

Inserts reported in "APL Heap Table" includes inserts made to worktables.

Updates and Update Detail Sections

The "Updates" report has two sections, "Updates" and "Data Only Locked Updates".

Updates

"Updates" reports updates to allpages-locked tables (APL) and data-only-locked tables (DOL) separately, depending on the type of update:

- APL Deferred
- APL Direct In-place
- APL Direct Cheap
- APL Direct Expensive
- DOL Deferred
- DOL Direct

The next section of the **sp_sysmon** report provides detailed information on updates to data-only-locked tables.

For more information on update types and their performance implications for allpages-locked tables, see "Update Operations" on page 8-34 of the *Performance and Tuning Guide*.

Data-Only-Locked Updates

This section reports on the "DOL Deferred" and "DOL Direct" updates to provide more detail.

- 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 noncontiguous 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

This section reports on deferred and direct deletes to allpages-locked tables separately. 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.

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 any of the tasks. If **housekeeper free write percent** is 1 or greater, the space reclamation task can be enabled separately, by setting **enable housekeeper GC** to 1, or disabled by setting it to 0.

Sample Output for Housekeeper Task Activity

```
Housekeeper Task Activity
-------------------------
                        per sec      per xact       count  % of total
                    ------------  ------------  ----------
Buffer Cache Washes
  Clean                    63.6           3.8       38163      96.7 %
  Dirty                     2.1           0.1        1283       3.3 %
                    ------------  ------------  ----------
Total Washes               65.7           3.9       39446

Garbage Collections         3.7           0.2        2230       n/a
Pages Processed in GC       0.0           0.0           1       n/a

Statistics Updates          3.7           0.2        2230       n/a
```

Buffer Cache Washes

This section reports:

- The number of buffers examined by the housekeeper
- The number that were found clean
- The number that were found dirty

The number found dirty does not indicate the number of writes started by the housekeeper; it also includes dirty buffers already in I/O.

The "Recovery Management" section of **sp_sysmon** reports how many times the housekeeper task was able to write all dirty buffers for a database. For more information, see "Recovery Management" on page 24-85 of the *Performance and Tuning Guide*.

Garbage Collections

This section reports on the number of times the housekeeper task woke up and 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 task succeeded in reclaiming unused space on the page. See "New Housekeeper Tasks" on page 18-22 for more information.

Statistics Updates

"Statistics Updates" reports on the number of times the housekeeper task woke up and checked to see if statistics needed to be written.

Index Management

The "Index Management" section now reports on these activities separately for data-only-locked tables:

- Index maintenance activity for updates and deletes
- Ascending and descending scans

Sample Output for Index Management

```
Index Management
----------------

  Nonclustered Maintenance     per sec   per xact    count  % of total
  -------------------------  ---------  ---------  -------  ----------
    Ins/Upd Requiring Maint       20.4        1.2    12269       n/a
      # of NC Ndx Maint            5.9        0.4     3535       n/a
      Avg NC Ndx Maint / Op        n/a        n/a  0.28812       n/a

    Deletes Requiring Maint       20.4        1.2    12259       n/a
      # of NC Ndx Maint            5.9        0.4     3514       n/a
      Avg NC Ndx Maint / Op        n/a        n/a  0.28665       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         7.3        0.4     4351       n/a
      # of DOL Ndx Maint          4.7        0.3     2812       n/a
      Avg DOL Ndx Maint / Op      n/a        n/a  0.64629       n/a

  Page Splits                     0.3        0.0      207       n/a
    Retries                       0.0        0.0        1       0.5 %
    Deadlocks                     0.0        0.0        0       0.0 %
    Empty Page Flushes            0.0        0.0        0       0.0 %
    Add Index Level               0.0        0.0        0       0.0 %
```

```
Page Shrinks                    0.0        0.0        0       n/a

Index Scans                  per sec   per xact    count  % of total
-------------------------  ---------  ---------  -------  ----------
  Ascending Scans              717.1       43.6   430258      90.6 %
  DOL Ascending Scans           74.3        4.5    44551       9.4 %
  Descending Scans               0.1        0.0       85       0.0 %
  DOL Descending Scans           0.0        0.0        6       0.0 %
                             ---------  ---------  -------
  Total Scans                  791.5       48.1   474900
```

Index Maintenance for Data-Only-Locked Tables

This section reports the number of times a data modification command required nonclustered index 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".

Scan Direction

The "Index Scans" section reports forward and backward scans by locking type:

- "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 "Queries That Use Mixed Ascending and Descending Order" on page 10-19.

Appendix

A System Tables

This appendix contains the following sections:

New and Changed System Tables

The following system tables are new or have new columns:

System Table	Description
sysindexes	Changed. Contains new columns to store space management property values.
syslocks	Changed. A new column stores the row number for row locks.
systabstats	New. Stores one row for each table, pl.us one row for each nonclustered index.
sysstatistics	New. Stores zero or more rows for each column in an index; can also stored one or more rows for unindexed columns.

The two new system tables, *systabstats* and *sysstatistics*, use datarow locking. All other system tables use allpages locking.

sysindexes

(all databases)

Description

sysindexes contains one row for each clustered index, one row for each nonclustered index, one row for each table that has no clustered index, and one row for each table that contains *text* or *image* columns.

Name	Datatype	Description
name	*sysname*	Index or table name
id	*int*	ID of table, or ID of table to which index belongs
indid	*smallint*	0 if a table; 1 if a clustered index on an allpages-locked table; >1 if a nonclustered index or a clustered index on a data-only-locked table; 255 if *text* or *image* object
doampg	*int*	Page number for the object allocation map of a table or clustered index
ioampg	*int*	Page number for the allocation map of a nonclustered index
oampgtrips	*int*	Number of times OAM pages cycle in the cache without being re-used, before being flushed
status2	*int*	Internal system status information (see Table A-1)
ipgtrips	*int*	Number of times index pages cycle in the cache, without being reused, before being flushed
first	*int*	Page number of the first data or leaf page
root	*int*	Page number of the root page if entry is an index; page number of the last page if entry is an unpartitioned table or text chain; unused if entry is a partitioned table (see "*syspartitions*" in Appendix A of *Adaptive Server Reference Manual*)
distribution	*int*	Formerly used to store the page number of the distribution page for an index.
usagecnt	*smallint*	Reserved
segment	*smallint*	Number of segment in which object resides

Name	Datatype	Description
status	*smallint*	Internal system status information (see Table A-2)
maxrowsperpage	*smallint*	Maximum number of rows per page
minlen	*smallint*	Minimum size of a row
maxlen	*smallint*	Maximum size of a row
maxirow	*smallint*	Maximum size of a non-leaf index row
keycnt	*smallint*	Number of keys for a clustered index on an allpages-locked table; number of keys, plus 1 for all other indexes
keys1	*varbinary(255)*	Description of key columns if entry is an index
keys2	*varbinary(255)*	Description of key columns if entry is an index
soid	*tinyint*	Sort order ID that the index was created with; 0 if there is no character data in the keys
csid	*tinyint*	Character set ID that the index was created with; 0 if there is no character data in the keys
base_partition	*int*	Partition number, incremented by **alter table...unpartition** commands
fill_factor	*smallint*	Fillfactor set for an index
res_page_gap	*smallint*	Value for the **reservepagegap** on a table
exp_rowsize	*smallint*	Expected size of data rows
keys3	*varbinary(255)*	Description of key columns if entry is an index

Table A-1 lists the bit representations for the *status2* column.

Table A-1: status2 bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Index supports foreign key constraint
2	0x2	Index supports primary key / unique declarative constraint
4	0x4	Index includes an IDENTITY column
8	0x8	Constraint name not specified
16	0x10	Large I / Os (prefetch) not enabled for table, index, or text chain

Table A-1: status2 bits in the sysindexes table (continued)

Decimal	Hex	Status
32	0x20	MRU cache strategy not enabled for table, index, or text chain
64	0x40	Ascending inserts turned on for the table
256	0x0100	Index is presorted and does not need to be copied to new extents
512	0x0200	Table is a data-only-locked table with a clustered index
8192	0x2000	Index on a data-only-locked table is suspect

Table A-2 lists the bit representations for the *status* column.

Table A-2: status bits in the sysindexes table

Decimal	Hex	Status
1	0x1	Abort current command or trigger if attempt to insert duplicate key
2	0x2	Unique index
4	0x4	Abort current command or trigger if attempt to insert duplicate row; always 0 for data-only-locked tables
16	0x10	Clustered index
64	0x40	Index allows duplicate rows, if an allpages-locked table; always 0 for data-only-locked tables
128	0x80	Sorted object; not set for tables without clustered indexes or for text objects
512	0x200	**sorted data** option used in **create index** statement
2048	0x800	Index on primary key
32768	0x8000	Suspect index; index was created under another sort order

Indexes

Unique clustered index on *id, indid*

Referenced by System Procedures

sp_cachestrategy, sp_checknames, sp_checkreswords, sp_chgattribute, sp_dropsegment, sp_estspace, sp_help, sp_helpconstraint, sp_helpindex, sp_helplog, sp_helpsegment, sp_indsuspect, sp_pkeys, sp_placeobject, sp_relimit, sp_rename, sp_spaceused, sp_special_columns, sp_statistics

syslocks

(*master* database only)

Description

syslocks contains information about active locks. It is built dynamically when queried by a user. No updates to *syslocks* are allowed.

Name	Datatype	Description
id	*int*	Table ID
dbid	*smallint*	Database ID
page	*int*	Page number
type	*smallint*	Type of lock (bit values for the *type* column are listed in Table A-3)
spid	*smallint*	ID of process that holds the lock
class	*char(30)*	Name of the cursor this lock is associated with, if any
fid	*smallint*	The family (coordinating process and its worker processes) to which the lock belongs. *fid* values are listed in Table A-4.
context	*tinyint*	Context type of lock request. *context* values are listed in Table A-5.
row	*smallint*	Row number

Table A-3 lists the bit representations for the *type* column.

Table A-3: type control bits in the syslocks table

Decimal	Hex	Status
1	0x1	Exclusive table lock
2	0x2	Shared table lock
3	0x3	Exclusive intent lock
4	0x4	Shared intent lock
5	0x5	Exclusive page lock
6	0x6	Shared page lock
7	0x7	Update page lock
8	0x8	Exclusive row lock
9	0x9	Shared row lock
10	0xA	Update row lock
11	0xB	Shared next key lock

Table A-3: type control bits in the syslocks table (continued)

Decimal	Hex	Status
256	0x100	Lock is blocking another process
512	0x200	Demand lock

Table A-4 lists the values for the *fid* column:

Table A-4: fid column values in the syslocks table

Value	Interpretation
0	The task represented by the *spid* is a single task executing a statement in serial.
Nonzero	The task (*spid*) holding the lock is a member of a family executing a statement in parallel. If the value is equal to the *spid*, it indicates that the task is the coordinating process in a family executing a query in parallel.

Table A-5 lists the values for the *context* column:

Table A-5: context column values in the syslocks table

Value	Interpretation
null	The task holding this lock is either executing a query in serial, or it is a query being executed in parallel in transaction isolation level 1.
0x1	The task holding the lock will hold the lock until the query is complete. A lock's context may be "Fam dur" when: • The lock is a table lock held as part of a parallel query. • The lock is held by a worker process at transaction isolation level 3. • The lock is held by a worker process in a parallel query and must be held for the duration of the transaction.
0x2	Range lock held by serializable read task
0x4	Infinity key lock
0x8	Lock acquired on an index pages of an allpages-locked table
0x10	Lock on a page or row acquired to delete a row
0x20	Address lock acquired on an index page during a shrink or split operation
0x40	Intent lock held by a transaction performing repeatable reads. Valid only for shared intent and exclusive intent locks on data-only locked tables.

Indexes

None

Referenced by System Procedures

sp_familylock, sp_lock

sysstatistics

(all databases)

Description

sysstatistics contains one or more rows for each indexed column on a user table. May also contain rows for unindexed column.

Name	Datatype	Description
statid	*smallint*	Reserved
id	*int*	Object ID of table
sequence	*int*	Sequence number if multiple rows are required for this set of statistics
moddate	*datetime*	Date this row was last modified
formatid	*tinyint*	Type of statistics represented by this row
usedcount	*tinyint*	Number of fields *c0* to *c79* used in this row
colidarray	*varbinary(100)*	An ordered list of column IDs
c0...c79	*varbinary(255)*	Statistical data

Indexes

Unique clustered index on *id, statid, colidarray, formatid, sequence*

Referenced by System Procedures

None

systabstats

(all databases)

Description

systabstats contains one row for each clustered index, one row for each nonclustered index, and one row for each table that has no clustered index.

Name	Datatype	Description
indid	*smallint*	0 if a table; 1 if a clustered index on an allpages-locked table; >1 if a nonclustered index or a clustered index on a data-only-locked table; 255 if *text* or *image* object
id	*int*	ID of table to which index belongs
activestatid	*smallint*	Reserved
indexheight	*smallint*	Height of the index; maintained if *indid* is greater than 1
leafcnt	*int*	Number of leaf pages in the index; maintained if *indid* is greater than 1
pagecnt	*int*	Number of pages in the table or index
rowcnt	*float*	Number of rows in the table; maintained for *indid* of 0 or 1
forwrowcnt	*float*	Number of forwarded rows; maintained for *indid* of 0 or 1
delrowcnt	*float*	Number of deleted rows
dpagecrcnt	*float*	Number of extent I/Os that need to be performed to read the entire table
ipagecrcnt	*float*	Number of extent I/Os that need to be performed to read the entire leaf level of a nonclustered index
drowcrcnt	*float*	Number of page I/Os that need to be performed to read an entire table
oamapgcnt	*int*	Number of OAM pages for the table, plus the number of allocation pages that store information about the table
extent0pgcnt	*int*	Count of pages that are on the same extent as the allocation page
datarowsize	*float*	Average size of the data row

Name	Datatype	Description
leafrowsize	*float*	Average size of a leaf row for nonclustered indexes and clustered indexes data-only-locked tables
status	*int*	Internal system status information (see Table A-6)
spare1	*int*	Reserved
spare2	*float*	Reserved
rslastoam	*int*	Last OAM page visited by a **reorg reclaim_space** or **reorg compact** command
rslastpage	*int*	Last data or leaf page visited by a **reorg reclaim_space** or **reorg compact** command
frlastoam	*int*	Last OAM page visited by the **reorg forwarded_rows** command
frlastpage	*int*	Last data page visited by the **reorg forwarded_rows** command
conopt_thld	*smallint*	Concurrency optimization threshold
spare3	*int*	Reserved
emptypgcnt	*int*	Number of empty pages in extents allocated to the table or index
spare4	*float*	Reserved

Table A-6 lists the bit representations for the *status* column:

Table A-6: status bits in the systabstats table

Decimal	Hex	Status
1	0x1	Statistics are the result of upgrade (not **update statistics**)

Indexes

Unique clustered index on *id, indid*

Referenced by System Procedures

sp_chgattribute

Tables with New Status Bits

The tables in this section have new status bits, but no other changes.

sysdatabases

Table A-7 lists the new bit representations for the *status2* column.

Table A-7: status2 bits in the sysdatabases table

Decimal	Hex	Status
512	0x0200	Database is in the process of being upgraded
1024	0x0400	Database brought online for standby access

sysobjects

Table A-8 lists the bit representations for the *sysstat2* column:

Table A-8: sysstat2 bits in the sysobjects table

Decimal	Hex	Status
8192	0x2000	Table uses allpages locking scheme
16384	0x4000	Table uses datapages locking scheme
32768	0x8000	Table uses datarows locking scheme
65536	0x10000	Table was created in a version 11.9 or later version of the server
131072	0x20000	Table has a clustered index
242144	0x40000	Object represents an Embedded SQL procedure

Entity Relationship Diagram for New System Tables

Figure A-1 shows the relationship between the two new system tables, *systabstats* and *sysstatistics*, and the *sysindexes* and *sysobjects* tables.

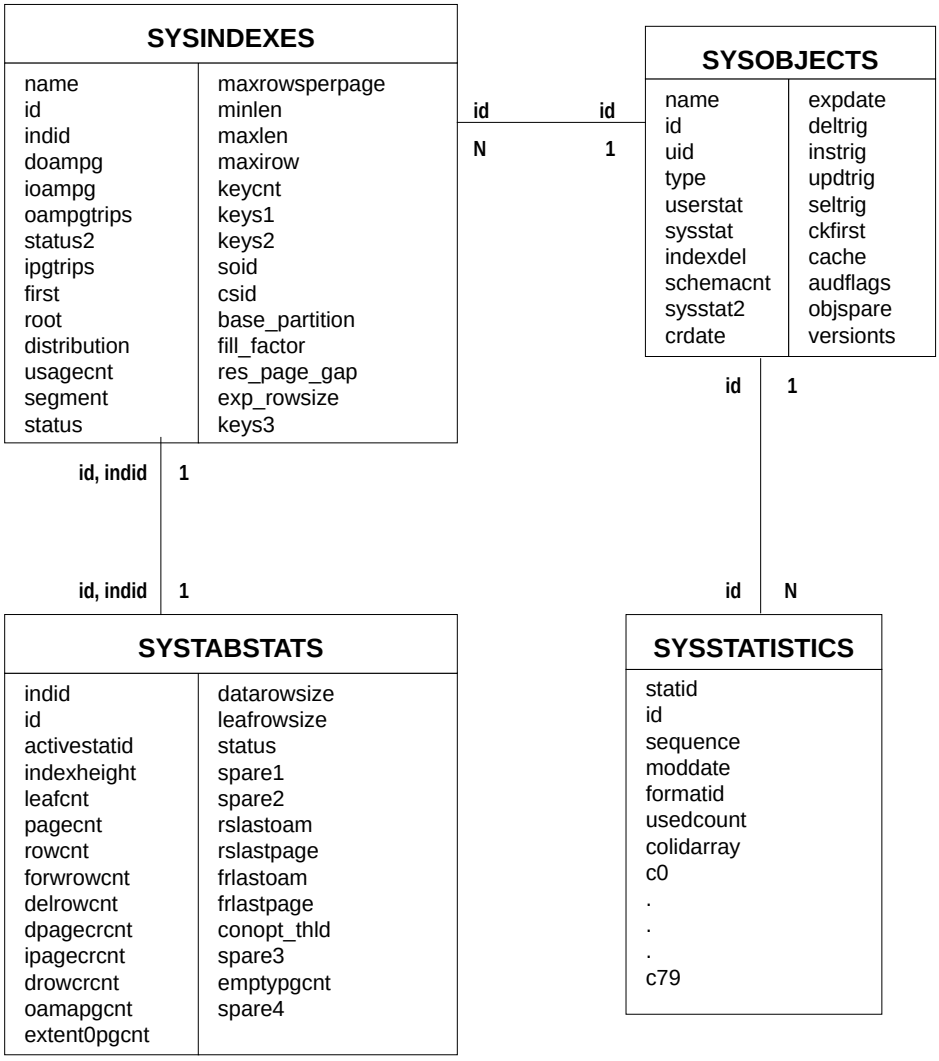

Figure A-1: Entity relationship diagram for new system tables

New *syblicenseslog* Table

syblicenseslog

(*master* database only)

Description

syblicenseslog contains one row for each update of the maximum number of licenses used in Adaptive Server per 24-hour period. *syblicenseslog* is updated every 24 hours. If Adaptive Server is shut down at any time, License Use Manager logs the number of licenses currently being used in *syblicenseslog* before the shutdown is complete. The 24 hour period restarts when you start Adaptive Server.

➤ ***Note***

syblicenseslog is not a system table. Its type is "U" and its object ID is greater than 100.

Columns

Name	Datatype	Description
status	*smallint*	Status of the maximum number of licenses used; one of the following: • 0 = number of licenses not exceeded • 1 = number of licenses is exceeded • -1 = housekeeper is unable to monitor number of licenses
logtime	*datetime*	Date and time the log was written
maxlicenses	*int*	Maximum number of licenses used during the 24-hour period

Indexes

None

Referenced by System Procedures

None

Glossary

allpages locking scheme

One of the three locking schemes in Adaptive Server. In allpages locking, locks are acquired on data and index pages. See also **datapages locking scheme** and **datarows locking scheme**.

blocking

Waiting for a lock; tasks that need to acquire a lock on a row, page, or table must wait, or block, if another process holds an incompatible lock.

cell

A pair of contiguous values in a **distribution** or **histogram**. Each step is associated with the set of rows having column values that fall between the upper and lower bound of the step. Also called a **step**.

child page pointer

A pointer in an index that points to the next lowest level of the index containing the key values. A child page pointer consists of a key and a row ID.

clustered index

For allpages-locked tables with clustered indexes, rows are stored in key order on the data pages, and the data pages are linked in the order of the key values. There is no leaf level in the index above the data level. For data-only locked tables, a clustered index is structurally a nonclustered index that directs the placement of data rows. Unlike a clustered index on an allpages-locked table, the sorted order of rows is not guaranteed. Only one clustered index can be designated per table.

data-only locking

A term used to denote both the **datapages locking scheme** and the **datarows locking scheme**. In these locking schemes, only the underlying data row or page is locked, and no index row locks or page locks are acquired. Index rows or pages are implicitly locked by locking the underlying data page.

data page cluster ratio

The percentage of page accesses that do not require an extra extent I/O when rows are accessed in index order. The database cluster ratio measures the ordering and density of data pages in extents with respect to data access via the given index.

data row cluster ratio

The percentage of row accesses for an index that do not require an extra logical or physical I/O when rows are accessed in index order. The data row cluster ratio is maintained for each clustered index on a data-only-locked table and for each nonclustered index. The data row cluster ratio is not maintained for a clustered index because the ratio is always 1, that is, 100 percent clustered, since rows are guaranteed to be in key order.

datapages locking scheme

One of the three locking schemes in Adaptive Server. In datapages locking, only data pages are locked. No page locks are acquired on index pages. See also **allpages locking scheme** and **datarows locking scheme**.

datarows locking scheme

One of the three locking schemes in Adaptive Server. In datarows locking, only data rows are locked. No row locks are acquired on index pages. See also **allpages locking scheme** and **datapages locking scheme**.

dense frequency count

A frequency count in which values are contiguous in the domain. For example, in the integer domain, the set of values 1, 2, 3 is dense. Compare to **sparse frequency count**.

density

A measure of the average number of duplicates per value in a column of a table. The query optimizer uses the **range cell density** or the **total density**, depending on the type of query.

distribution

An ordered set of n+1 values from a column of a table, obtained by sorting all the values and then selecting the first value and every subsequent (*row_count* / *n*th) value (where *row_count* is the number of rows in the table).

distribution page

A page associated with an index that was used to store statistics about data distribution in pre-11.9 versions of SQL Server and Adaptive Server.

exclusive row lock

A type of lock acquired on a data row during a write operation. It does not allow other transactions to acquire exclusive, update, or shared locks on the row.

expected row size

A user-defined average for the expected size of a data row. Specifying the expected row size for a table reserves space on the data page for rows that grow in length after they are inserted. This helps to avoid **row forwarding**.

extent allocation

A method of allocating new space for a table or an index. In most cases, new space is allocated one page at a time. Commands that need to allocate large amounts of space can perform extent allocation, allocating eight pages at a time.

fixed row ID

A method of ensuring that the **row ID (RID)** associated with a row does not change. The **offset table** of the row may change because of row movement within the page, but the row number itself does not change. If row expansion requires a row to move, the row migrates to a new page and the **forwarding address** is stored in the original location of the row. The row ID does not change for the life of the row, except when clustered indexes are created or when the **reorg rebuild** command is run. Row IDs are reused after a row is physically deleted from a page.

forcing

Including options in a query or session to override the optimizer's choice for a query plan.

forwarded row

A row in a data-only-locked table that has been migrated to a new data page, because of a change in the length of the row. The previous location of the row stores a pointer to the forwarded row's location.

forwarding address

The new location of a row. The forwarding address is in the original location of the row.

frequency cell

A cell in a **histogram** that represents a **frequency count**. The weight of a frequency cell indicates the percentage of columns that match the value for the cell. Compare to **range cell**.

frequency count

A method of modeling a data distribution with a low number of values in the domain. For example, a gender column can be modeled by two values, "M" and "F", which have a count of about 0.48 and 0.52, respectively.

hash buckets

Structures used to manage certain shared resources, such as locks.

histogram

A set of cells in which each cell has a weight. Each cell has an upper limit, a lower bound, and a float value between 0 and 1 representing the percentage of rows within the range. Adaptive Server statistics for column values are stored as histograms in the *sysstatistics* system table. These statistics are used by the query optimizer.

home page

The original location of a row. If the row has been moved by **row forwarding**, the **home page** contains the **forwarding address** of the row.

index page cluster ratio

The percentage of index leaf page accesses via the page chain that do not require extra extent I/O. The index page cluster ratio is maintained for clustered indexes on data-only-locked tables and for nonclustered indexes.

latch

A lightweight, non-transactional synchronization mechanism held for a very short duration to ensure the physical consistency of the page.

lock promotion

Promotion of the row or page locks acquired by a scan to a table lock. Lock promotion takes place when a scan has exceeded the user-configurable setting for the lock promotion threshold. Locks are promoted from row locks to table locks, or page locks to table locks.

magic values

Values that are substituted or used as defaults when actual values are not known. For example, the optimizer uses magic numbers for the selectivity of a value in a parameter that cannot be determined when the query is optimized.

minor column

A non-leading column of a composite index. For an index on (A, B, C), B and C are minor columns.

next-key locking

A strategy to protect repeatable read transactions from reading **phantom rows**.

OAM scan

A method of accessing data by first reading the OAM pages for a table, then the allocation pages, and finally the data pages where the data is stored.

offset table

A set of fields at the end of data and index pages that store pointers to the byte location of the rows on the page. Indexes on allpages-locked tables do not have offset tables; all other indexes and all data pages do.

positioned delete

A **delete** performed through a cursor, using the **where current of** clause. Compare to **searched delete**.

positioned update

An update performed through a cursor, using the **where current of** clause. Compare to **searched update**.

range cell

A cell of a **histogram** which represents a range of values. The weight of the cell represents the percentage of column values that are greater than the lower bound of the cell, and less than or equal to the upper bound. Compare to **frequency cell**.

range cell density

A measure of duplicates in a column or set of columns which has some duplicate values removed when more than one distribution cell is spanned. Compare to **total density**.

range lock

A lock acquired by a range query at transaction isolation level 3, for the purpose of preventing phantom rows. Range locks prevent inserts into the range by other users until the repeatable read transaction completes. Range locks are used on data-only locked tables.

repeatable read

A query in which the results are not affected by changes made by other transactions. The transaction holds locks on the pages or rows it has read until end of transaction to ensure other transactions do not update the pages or rows. Also known as transaction isolation level 2. This level does not provide phantom protection, which is provided at transaction isolation level 3.

reserve page gap

A ratio to be used during large scale allocations performed by bulk copy, **create index** and **select into**. Setting a reserved page gap value allows future page allocations for forwarded rows and index page splits to be made close to the original location of the data.

RID

See **row ID (RID)**.

rollback record

A log record generated by Adaptive Server to describe the undoing of an earlier change of the transaction. The rollback record contains the address of the log record that is being undone. Rollback records are redo-only log records. They are never undone.

row ID (RID)

A unique identifier for the data row. A row ID consists of the page number where the row is located and the row number within the page.

row forwarding

Movement of a row to another page. Row forwarding takes place when a row is updated so that it no longer fits on the page. A pointer to the **forwarded row** is maintained at the original row location. The row ID of the row does not change.

searched delete

A delete performed while a cursor is active, but not through the cursor, that is, not using the **where current of** clause. Compare to **positioned delete**.

searched update

An update performed while a cursor is active, but not through the cursor, that is, not using the **where current of** clause. Compare to **positioned update**.

serializable read

A query within a transaction that always returns the same set of rows. Prevents phantoms, rows that appear or disappear from the result set. Also called transaction isolation level 3.

shared row lock

A type of lock acquired on a data row for a read operation. It does not allow other transactions to acquire exclusive locks on the row, but allows them to acquire shared locks.

space management property

One of the properties of a table or index that helps manage how space for the table is allocated or used: **fillfactor**, **max_rows_per_page**, **exp_row_size** and **reservepagegap**. Space management properties are specified for tables with **create table** or **alter table** and for indexes with **create index** or the **alter table** commands that generate indexes to enforce unique or primary key constraints. Some of the space management properties can be changed with **sp_chgattribute**.

space reclamation

The process of recovering space from data-only locked tables after deletes and updates that shrink rows. Space reclamation is performed by tasks inserting or updating rows on a page, by the housekeeper, and by the **reorg** command.

sparse frequency count

A **frequency count** in which values are not contiguous in the domain. For example, the set of values 1, 10, 100 would be sparse in the integer domain, but 1, 2,3 would not be sparse. Compare to **dense frequency count**.

standby access server

A server that is an exact duplicate of an Adaptive Server production server. A standby server is created by loading transaction log dumps from the production server to the standby server as the log dumps are made. Standby servers can provide emergency coverage in case the main server fails, or they can be read-only servers used for reporting or decision support.

step

A pair of contiguous values in a **distribution** or **histogram**. Each step is associated with the set of rows having column values that fall between the upper and lower bound of the step. Also called a **cell**.

total density

A measure of duplicates in a column or set of columns which represents all duplicates even when more than one histogram cell is spanned. Compare to **range cell density**.

update row lock

A type of lock acquired on a data row during the scanning phase of an update or delete operation. It does not allow other transactions to acquire update or exclusive locks on the row but allows them to acquire shared locks.

Index

D

E

F

H

I

J

K

L

P

Q

R

S

T

U

V

W