SYBASE®
An SAP® Company

Performance and Tuning Series:
Improving Performance with Statistical Analysis

Adaptive Server® Enterprise

15.7

DOCUMENT ID: DC00976-01-1570-01

LAST REVISED: September 2011

Contents

CHAPTER 1

Using the *set statistics* Commands

The set command includes the statistics parameter, which includes options that display performance statistics. The default setting for all statistics options is off.

set command syntax

The syntax for set statistics is:

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

You can issue a single parameter:

```
set statistics io on
```

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

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

Using simulated statistics

The optdiag utility allows you to load simulated statistics and perform query diagnosis using those statistics. You can load simulated statistics even for empty tables, which lets you perform tuning diagnostics even in small databases that contains only tables and indexes. Simulated statistics do not overwrite existing statistics when they are loaded.

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

```
set statistics simulate on
```

See "Using simulated statistics" on page 44.

Checking subquery cache performance

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

Checking compile and execute time

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

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

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

The meaning of this output is:

- Parse and Compile Time – the number of CPU ticks taken to parse, optimize, and compile the query. See below for information on converting ticks to milliseconds.
- Adaptive Server® cpu time – the CPU time in milliseconds.

- Execution Time – the number of CPU ticks taken to execute the query.
- Adaptive Server cpu time – the number of CPU ticks taken to execute the query, converted to milliseconds.
- Adaptive Server elapsed time – the difference in milliseconds between the time the command started and the current time, taken from the operating system clock.

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

Note If a query is complex and takes a long time to run, check if Execution Time is high because of a suboptimal plan or that Parse and Compile Time is high because of optimization performance.

Converting ticks to milliseconds

To convert ticks to milliseconds, use:

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

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

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

See Chapter 5, “Setting Configuration Parameters” in *System Administration Guide: Volume 1*.

Reporting physical and logical I/O statistics

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

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

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

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

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

Total actual I/O cost value

If resource limits are enabled, statistics io prints the "Total actual I/O cost" line. Adaptive Server reports the total actual I/O as a unitless number. The formula for determining the cost of a query is:

Cost = All physical IOs X 25 + All logical IOs X 2 + CPU cost X 0.1

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

For example:

```
Table: sysmessages scan count 1, logical reads:
(regular=454 apf=0 total=454), physical reads:
(regular=441 apf=0 total=441), apf IOs used=10
Total actual I/O cost for this command: 11934
```

That is: 441*25 + 454*2 + 10*0.1 = 11934

Table 1-1 describes "regular" and "apf" reads.

Statistics for writes

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

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

Statistics for reads

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

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

Table 1-1: statistics io output for reads

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

Sample output with and without an index

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

```
select title
```

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

statistics io without an index

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

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

This output shows that:

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

statistics io with an index

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

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

The output shows that:

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

statistics io output for cursors

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

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


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

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

Scan count

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

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

See Chapter 2, "Using showplan," in *Performance and Tuning Series: Query Processing*.

Queries reporting a scan count of 1

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

- A point query:

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

- A range query:

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

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

Queries reporting a scan count of more than 1

Examples of queries that return larger scan count values are:

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

  The following query reports a scan count of 2 if the titles table has indexes on title_id and another on pub_id:

  ```
  select title_id
  from titles
      where title_id = "T55522"
      or pub_id  = "P988"
  ```

```
Table: titles scan count 2,logical reads: (regular=149 apf=0
total=149), physical reads: (regular=63 apf=80 total=143), apf IOs
used=80
Table: Worktable1  scan count 1, logical reads: (regular=172 apf=0
total=172), physical reads: (regular=0 apf=0 total=0), apf IOs
```

The I/O for the worktable is also reported.

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

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

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

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

Queries reporting scan count of 0

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

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

Relationship between physical and logical reads

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

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

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

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

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

Physical reads and large I/O

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

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

Reads and writes on worktables

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

Effects of caching on reads

If you are testing a query and checking the query’s I/O, and you execute the same query a second time, you may get surprising physical read values, especially if the query uses least recently used (LRU) replacement strategy.

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

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

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

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

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

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

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

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

- Changing the cache binding for the object:

  - If a table or index is bound to a cache, unbind it, and rebind it.
  - If a table or index is not bound to a cache, bind it to any cache available, then unbind it.

  You must have at least one user-defined cache to use this option.

- If you do not have any user-defined caches, you can execute a sufficient number of queries on other tables so that the objects of interest are flushed from cache. If the cache is very large, this can be time-consuming.

- Restarting the server.

For more information on testing and cache performance, see Chapter 5, "Memory Use and Performance" in *Performance and Tuning Series: Basics*.

statistics io **and merge joins**

statistics io output does not include sort costs for merge joins. If you enable allow resource limits, the sort cost is not reported in the “Total estimated I/O cost” and “Total actual I/O cost” statistics.

Analyzing queries with *set statistics plancost*

set statistics plancost simplifies query analysis. It displays the estimated values for logical I/O, physical I/O, and row counts compared to the actual ones evaluated at each operator, and reports on CPU and sort buffer cost.

See “set statistics plancost” in Chapter 6, Ensuring Stability and Performance” in the *Migration Guide*.

set statistics plancost example

set statistics plancost simplifies query analysis by displaying estimated values for logical I/O, physical I/O, and row counts compared to the actual ones evaluated at each operator, and reports on CPU and sort buffer cost.

```
set statistics plancost on
```

Use plancost to compare a query plan’s estimated and actual costs. plancost may also help you diagnose query performance problems. Table 1-2 describes plancost’s actual and estimated output.

Table 1-2: Estimated and actual costs

For each operator, plancost shows	For this operation
`el:` – estimated `l:` – actual	Logical I/O
`ep: – estimated` `p:` – actual	Physical I/O
`er:` – estimated `r:` – actual	Row counts
`cpu:` – actual	CPU counts

Execution operators that perform sort- or hash-based operations report the number of private buffers used for that operation ("`bufct:`", not shown in the example below). plancost may display a subset of costs because not all these quantities apply to all operators. Use the subset cost to verify whether the optimizer's estimates are valid for sub-optimal query plans.

For example, plancost issues the following output when you run a join query on the authors, titleauthor, and titles tables:

```
select A.au_fname, A.au_lname, T.title
from authors A, titleauthor TA, titles T
where A.au_id = TA.au_id and T.title_id = TA.title_id
==================== Lava Operator Tree ====================
                                            Emit
                                            (VA = 6)
                                            r:25 er:342
                                            cpu: 0
                                    /
                                    MergeJoin
                                    Inner Join
                                    (VA = 5)
                                    r:25 er:342
                          /                            \
                        Sort                           IndexScan
                        (VA = 3)                       titles_indx (T)
                        r:25 er:25                     (VA = 4)
                        l:6 el:6                       r:18 er:18
                        p:0 ep:0                       l:2 el:3
                        cpu: 0 bufct: 24               p:0 ep:3
                 /
                MergeJoin
                Inner Join
                (VA = 2)
                r:25 er:25
          /                                  \
IndexScan                                IndexScan
auidind (TA)                             authors_indx (A)
(VA = 0)                                 (VA = 1)
r:25 er:25                               r:23 er:23
l:1 el:2                                 l:1 el:2
p:0 ep:2                                 p:0 ep:2
```

The query optimizer produces the estimated numbers; the actual numbers are the result of the query execution.

This output shows the bottom-left IndexScan operator (on the titleauthors table) has estimated (`er:`) and actual (`r:`) row counts of 25, which means the optimizer's estimate was correct. However, the row count estimates for the top MergeJoin (`VA = 5`) are incorrect: the query processor's estimate is 342 but the actual row count is 25.

You may be able to more accurately controle the query processor's estimates by keeping the statistics up to date, or by increasing the number of histogram steps. Use set option show_missing_stats on to verify whether join columns have histograms. You may be able to improve the query processor's estimates by creating histograms, if none exist.

If an estimated row count is 25 but the actual row count is 30, this is not necessarily an indication that the query processor's estimates are incorrect. When comparing estimated and actual values, look for "order-of-magnitude differences" (such as 25 versus 342 in the example above).

The query processor displays the name of the index (not the table name) for `IndexScan` operator nodes. To determine which table is associated with an operator node:

- The index name may uniquely identify the table.
- The operator node output includes a correlation name if the correlation name was included in the query. For example, in the above output, the "`(TA)`" in the `IndexScan` operator, refers to "`titleauthor TA`" from the SQL query.

The query tree and the showplan output include "`(VA=`***`n`***`)`", where *n* = 0, 1, 2, and so on, and uniquely identifies each operator node.

CHAPTER 2 Statistics Tables and Displaying Statistics with *optdiag*

This chapter explains how statistics are stored and displayed.

System tables that store statistics

The systabstats and sysstatistics tables store statistics for all tables, indexes, and any unindexed columns for which you have explicitly created statistics:

- systabstats stores information about the table or index as an object, and is updated by query processing, data definition language, and update statistics commands
- sysstatistics stores information about the values in a specific column, and is updated by data definition language and update statistics commands

See "Effects of SQL commands on statistics" on page 50. For complete information about all system tables, including these, see the *Reference Manual: Tables*

systabstats **table**

The systabstats table contains basic statistics for tables and indexes, including:

- Number of data pages for a table, or the number of leaf level pages for an index
- Number of rows in the table
- Height of the index
- Average length of data rows and leaf rows
- Number of forwarded and deleted rows
- Number of empty pages
- Statistics to increase the accuracy of I/O cost estimates, including cluster ratios, the number of pages that share an extent with an allocation page, and the number of OAM and allocation pages used for the object
- Stopping points for the reorg command so that it can resume processing

systabstats contains one row for each clustered index, one row for each nonclustered index, one row for each table without a clustered index, and one row for each partition. The storage for clustered index information depends on the locking scheme for the table:

- For data-only-locked tables, systabstats stores an additional row for a clustered index.
- For allpages-locked tables, 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.

sysstatistics **table**

The sysstatistics table stores one or more rows for each indexed column on a user table; it also stores 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 25.

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. To generate a histogram, run update statistics after inserting data.

See "Histogram displays" on page 30.

Viewing statistics with the *optdiag* utility

The optdiag utility displays statistics from the systabstats and sysstatistics tables. You can also use optdiag to update sysstatistics information. Only a system administrator can run optdiag.

optdiag syntax

The syntax for optdiag is:

optdiag
[binary] [simulate] statistics {-i *input_file* |
database[.*owner*[.*table*[.*partition* [.*column*]]]]
[-o *output_file*]} [-U *user_name*] [-P *password*] [-I *interfaces_file*]
[-S *server*] [-v] [-h] [-s] [-T *flag_value*] [-z *language*]
[-J *client_charset*] [-a *display_charset*]

Use optdiag to display statistics for an entire database, for a single table and its indexes and columns, or for a particular column.

For example, display statistics for all user tables in the pubtune database, placing the output in the *pubtune.opt* file, use:

```
optdiag statistics pubtune -Usa -Ppasswd -o pubtune.opt
```

To display statistics for the titles table and for any indexes on the table, use:

```
optdiag statistics pubtune..titles -Usa -Ppasswd -o titles.opt
```

Running optdiag from a sliced table

For optdiag output from a slice table from an Adaptive Server® version earlier than 15.0, data is loaded according to the destination table's schema:

- The destination table is a single-partition table – optdiag loads the table using the standard method, ignoring the "Pages in largest partition" field.
- The destination table is a multipartition table and the "Pages in the largest partition" field is empty – optdiag loads the statistics into the first partition.
- The "Pages in largest partition" field is not empty – optdiag uses the value in this field to divide the values proportionally for the "Data page count," and "Data row count" fields into all partitions. optdiag always loads column-level statistics into the global statistics for columns.

See the *Utility Guide* for complete information about optdiag. The following sections provide information about optdiag output.

optdiag header information

After printing version information, 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 2-1 describes the output.

Table 2-1: Table and column information

Row Label	Information Provided
Server name	The name of the server, as stored in the *@@servername* variable. You must use sp_addserver, and restart the server for the server name to be available in the variable.
Specified database	Database name given on the optdiag command line.
Specified table owner	Table owner given on the optdiag command line.
Specified table	Table name given on the optdiag command line.
Specified column	Column name given on the optdiag command line.

Table statistics

This is sample optdiag output for table statistics:

```
Table owner:                        "dbo"
Table name:                         "authors"

Statistics for table:               "authors"

     Partition count:               3

Statistics for partition:           "authors_1376004902"
     Data page count:               74
     Empty data page count:         0
     Data row count:                1666.0000000000000000
     Forwarded row count:           0.0000000000000000
     Deleted row count:             0.0000000000000000
     Data page CR count:            10.0000000000000000
     OAM + allocation page count:   3
     First extent data pages:       0
     Data row size:                 85.2623049219687914
     Parallel join degree:          0.0000000000000000
     Unused page count:             5
     OAM page count:                1

  Derived statistics:
     Data page cluster ratio:       1.0000000000000000
     Space utilization:             0.9521597490347491
     Large I/O efficiency:          1.0000000000000000
```

Table 2-2: Table statistics

Row label	Information provided
Table owner	Name of the table owner. You can omit owner names on the command line by specifying *dbname..tablename*. If multiple tables have the same name and different owners, optdiag prints information for each table with that name.
Table name	Name of the table.
Statistics for table	Name of the table for which statistics are printed.
Partition count	Number of partitions.
Statistics for partition	Name of the partition for which the statistics are shown.
Data page count	Number of data pages in the table.
Empty data page count	Count of pages that have only deleted rows.
Data row count	Number of data rows in the table.
Forwarded row count	Number of forwarded rows in the table. This value is always 0 for an allpages-locked table.
Deleted row count	Number of rows that have been deleted from the table. These are committed deletes where the space has not been reclaimed by one of the functions that clears deleted rows. This value is always 0 for an allpages-locked table.
Data page CR count	A counter used to derive the data page cluster ratio. Computes the data page cluster ratio, which can help determine the effectiveness of large I/O for table scans and range scans. This value is only updated when you run update statistics
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.
Parallel join degree	An integer value that indicates the degree of parallelism used for a nested-loop join.
Unused page count	Number of unused pages in the extent.
OAM page count	Number of OAM pages.
Derived statistics	Group of statistics for which optdiag derives information.
Data page cluster ratio	See "Data page cluster ratio", below.

Row label	Information provided
Space utilization	See "Space utilization" below.
Large I/O efficiency	See "Large I/O efficiency" below.

Table-level derived statistics

"Derived statistics" report the statistics for "Data page cluster ration," "Space utilization," and "Large I/O efficiency" which are derived from the "Data Page CR count" and data page count.

Data page cluster ratio

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

For data-only-locked tables, the data page cluster ratio measures how well the pages are packed on extents. A cluster ratio of 1.0 indicates complete packing of extents. A low data page cluster ratio indicates that there are empty pages in extents that are allocated to the table.

Space utilization

Space utilization uses the average row size and number of rows to compute the expected minimum number of data pages, and compares the expected minimum to the current number of pages. If space utilization is low, run reorg rebuild on the table or drop and recreate the clustered index to reduce the amount of empty space on data pages, and the number of empty pages in extents allocated to the table.

If you use space management properties such as fillfactor or reservepagegap, the empty space that is left for additional rows on data pages of a table with a clustered index and the number of empty pages left in extents for the table affects the space utilization value.

If statistics have not been updated recently and the average row size has changed or the number of rows and pages are inaccurate, space utilization may report values greater than 1.0.

Large I/O efficiency

"Large I/O efficiency" estimates the number of useful pages brought in by each large I/O. For example, if the value of "Large I/O efficiency is.5, a 16K I/O returns, on average, 4 2K pages that are needed for the query, and 4 other pages, either empty pages or pages that share the extent due to lack of clustering. Low values are an indication that re-creating the clustered index or running reorg rebuild on the table may improve I/O performance.

Index statistics

This sample output is printed for each nonclustered index, a clustered index on a data-only-locked table, and for a clustered index for an all-pages locked table. Information for clustered indexes on allpages-locked tables is reported as part of the table statistics. Table 2-3 on page 23 describes the output.

```
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
```

Note Parallel join degree, Unused page count, and OAM page count, which are not included in this sample, appear only for all-pages locked tables with a clustered index.

Table 2-3: 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-level derived statistics” on page 23.
Index page CR count	A counter used to compute the index page cluster ratio. See “Index-level derived statistics” on page 23.
Data row CR count	A counter used to compute the data row cluster ratio See “Index-level derived statistics” on page 23.
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	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.

Index-level derived statistics

The derived statistics in the index-level section are based on the “CR count” values shown in “Index statistics” on page 22.

Data page cluster ratio

The data page cluster ratio is used to compute the effectiveness of large I/O when an index is used to access the data pages. If the table is perfectly clustered with respect to the index, the cluster ratio is 1.0. Data page cluster ratios can vary widely. They are often high for some indexes, and very low for others.

Index page cluster ratio

The index page cluster ratio is used to estimate the cost of large I/O for queries that need to read a large number of leaf-level pages from nonclustered indexes or clustered indexes on data-only-locked tables. Some examples of such queries are covered index scans and range queries that read a large number of rows.

On newly created indexes, the “Index page cluster ratio” is 1.0, or very close to 1.0, indicating optimal clustering of index leaf pages on extents. As index pages are split and new pages are allocated from additional extents, the ratio drops. A very low percentage may indicate that dropping and re-creating the index, or running reorg rebuild on it, 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 the index to access the data pages.

Space utilization for an index

Space utilization uses the average row size and number of rows to compute the expected minimum size of leaf-level index pages and compares it to the current number of leaf pages.

If space utilization is low, running reorg rebuild on an index, or dropping and re-creating it, can reduce the amount of empty space on index pages, and the number of empty pages in extents allocated to the index.

If you are using space management properties such as fillfactor or reservepagegap, the empty space that is left for additional rows on leaf pages, and the number of empty pages left in extents for the index affects space utilization.

If statistics have not been updated recently and the average row size has changed or the number of rows and pages are inaccurate, space utilization may report values greater than 1.0.

Large I/O efficiency for an index

Large I/O efficiency estimates the number of useful pages brought in by each large I/O. For examples, if the value is .5, a 16K I/O returns, on average, 4 2K pages needed for the query, and 4 other pages, either empty pages or pages that share the extent due to lack of clustering.

Low values are an indication that re-creating indexes or running reorg rebuild may improve I/O performance.

Column statistics

optdiag column-level statistics include:

- Statistics giving the density and selectivity of columns. If an index includes more than one column, optdiag prints the information described in Table 2-4 for each prefix subset of the index keys. If statistics are created using update statistics with a column name list, density statistics are stored for each prefix subset in the column list.
- A histogram, if the table contains one or more rows of data at the time the index is created or if you run update statistics. 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, these columns in the authors table 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 statistics for the city column in the authors table:

```
Statistics for column:                    "au_lname"
Derived statistics from local partitions.
Last update of column statistics:          Apr 8 2008 9:29:07:706PM

     Range cell density:                   0.0005507993510047
     Total density:                        0.0005507993510047
     Range selectivity:                    default used (0.33)
     In between selectivity:               default used (0.25)
     Unique range values:                  0.0005215561314205
     Unique total values:                  0.0005215561314205
     Average column width:                  6.7988000000000000
```

Table 2-4: 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 from a distribution page from a version of Adaptive Server earlier than 11.9. 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, run update statistics .
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 28.
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 28.
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. See “Range and in-between selectivity values” below.

Row label	Information provided
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. See "Range and in-between selectivity values" below.
Unique range values	Number of unique values in a column, excluding the frequency cells.
Unique total values	Total number of unique values.
Average column width	Average of all column widths from the table.

Range cell and total density values

Adaptive Server stores two values for the density of column values:

- "Range cell density" measures the duplicate values only for range cells.

  If there are any frequency cells for the column, they are eliminated from the computation for the range-cell density.

  If there are only frequency cells for the column, and no range cells, the range-cell density is 0.

  See "Understanding histogram output" on page 31 for information on range and frequency cells.

- "Total density" measures the duplicate values for all columns, those represented by both range cells and frequency cells.

Using two separate values improves the optimizer's estimates of the number of rows to be returned:

- If a search argument matches the value of a frequency cell, the fraction of rows represented by the weight of the frequency cell is returned.
- If a search argument falls within a range cell, the range-cell density and the weight of the range cell are used to estimate the number of rows to be returned.

For joins, the optimizer bases its estimates on the average number of rows to be returned for each scan of the table, so the total density, which measures the average number of duplicates for all values in the column, provides the best estimate. The total density is also used for equality arguments when the value of the search argument is not known when the query is optimized.

See "Range and in-between selectivity values" on page 30.

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 group:             "pub_id", "type"
Last update of column statistics:         Apr 8 2008 9:22:40:963AM

     Range cell density:            0.0000000362887320
     Total density:                 0.0000000362887320
     Range selectivity:             default used (0.33)
     In between selectivity:        default used (0.25)
     Unique range values:           0.0000000160149449
     Unique total values:           0.0000000160149449
     Average column width:          default used (8.00)

Statistics for column group:       "pub_id", "type", "pubdate"
Last update of column statistics:   Apr 8 2008 9:22:40:963AM

     Range cell density:            0.0000000358937986
     Total density:                 0.0000000358937986


     Range selectivity:             default used (0.33)
     In between selectivity:        default used (0.25)
     Unique range values:           0.0000000158004305
     Unique total values:           0.0000000158004305
     Average column width:          2.0000000000000000
```

With 5000 rows in the table, the increasing precision of the optimizer's estimates of rows to be returned depends on the number of search arguments used in the query:

- An equality search argument on only pub_id results in the estimate that 0.0335391029690461 * 5000 rows, or 168 rows, will be returned.
- Equality search arguments for all three columns result in the estimate that 0.0002011791956201 * 5000 rows, or only 1 row, will be returned.

This increasing level of accuracy as more search arguments are evaluated can greatly improve the optimization of many queries.

Range and in-between selectivity values

optdiag prints the default values for range and in-between selectivity, or the values that have been set for these selectivities in an earlier optdiag session. These values are used for range queries when search arguments are not known when the query is optimized.

For equality search arguments whose value is not known, the total density is used as the default.

Search arguments cannot be known at optimization time for:

- Stored procedures that set variables within a procedure
- Queries in batches that set variables for search arguments within a batch

Approximations for values 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 41 for information on using optdiag to supply selectivity values.

Histogram displays

Histograms store information about the distribution of values in a column. Table 2-5 shows the commands that create and update histograms and which columns are affected.

Table 2-5: 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
Sampling percent:                   0
```

```
    Out of range histogram adjustment:   DEFAULT
```

optdiag first prints summary data about the histogram, as shown in Table 2-6.

Table 2-6: Histogram summary statistics

Row label	Information provided
Histogram for column	Name of the column.
Column datatype	Datatype of the column, including the length, precision and scale, if appropriate for the datatype.
Requested step count	Number of steps requested for the column.
Actual step count	Number of steps generated for the column. This number can be fewer than the requested number of steps if the number of distinct values in the column is smaller than the requested number of steps.
Sampling percent	Percentage of table data sampled to produce the histogram. Value range is from 0 – 100.
Out of range histogram adjustment	Indicates whether the column's histogram is to be adjusted to assign a selectivity value for out-of-bounds search arguments on the column. Value is one of on, off, or default. This row appears along with histogram information only for global column statistics, not at the partition level.

Histogram output is printed in columns, as described in Table 2-7.

Table 2-7: 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 depend on whether the cell represents a range cell or a frequency call. No heading prints for this column.
Value	Upper boundary of the values represented by a range cell, or the value represented by a frequency count.

Understanding histogram output

A histogram is a set of cells in which each cell is assigned a weight. Each cell has an upper bound and a lower bound, which are distinct column values. 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 equiheight 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 approximately 5 percent of the table:

```
Step     Weight                Value

   1     0.00000000     <=     "APO Miami\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 that is less than the minimum column value (in the output above, “APO Miami”), 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 above 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:

lowerbound < (values for cell) <= upper bound

In optdiag output, the lower bound is the value of the previous step, and the upper bound is the value of the current step.

For example, in the histogram above, step 4 includes Charlotte (the upper bound), but excludes Boston (the lower bound). The weight for this step is .0540, indicating that 5.4 percent of the table matches the query clause:

```
where city > Boston and city <= "Charlotte"
```

The operator column in the optdiag histogram output shows the <= operator. Different operators are used for histograms with highly duplicated values.

Histograms for columns with highly duplicated values

Histograms for columns with highly duplicated values look very different from histograms for columns with a large number of discrete values. In histograms for columns with highly duplicated values, a single cell, called a **frequency cell**, represents the duplicated value.

The weight of the frequency cell shows the percentage of columns that have matching values.

Histogram output for frequency cells varies, depending on whether the column values represent one of the following:

- A dense frequency count, where values in the column are contiguous in the domain. For example, 1, 2, 3 are contiguous integer values.
- A sparse frequency count, where the domain of possible values contains values not represented by the discrete set of values in the table.
- A mix of dense and sparse frequency counts.

Histogram output for some columns includes a mix of frequency cells and range cells.

Histograms for dense frequency counts

The following output shows the histogram for a column that has 6 distinct integer values, 1 – 6, and some null values:

```
Step     Weight               Value

   1     0.13043478     <=     1
   2     0.04347826     <=     1
   3     0.17391305     <=     2
   4     0.30434781     <=     3
   5     0.13043478     <=     4
   6     0.17391305     <=     5
   7     0.04347826      <=     6
```

The histogram above shows a dense frequency count, because all the values for the column are contiguous integers.

The first cell represents null values (described in section below). Since there are null values, the weight for this cell represents the percentage of null values in the column.

The “Value” column for the first step displays the minimum column value in the table and the < operator.

Histograms for sparse frequency counts

In a histogram representing a column with a sparse frequency count, the highly duplicated values are represented by a step showing the discrete values with the = operator and the weight for the cell.

Preceding each step, there is a step with a weight of 0.0, the same value, and the < operator, indicating that there are no rows in the table with intervening values. For columns with null values, the first step has a nonzero weight if there are null values in the table.

The following histogram represents the type column of the titles table. Since there are only 9 distinct types, they are represented by 18 steps.

```
Step      Weight                   Value

    1     0.00000000        <        "UNDECIDED   "
    2     0.11500000        =        "UNDECIDED  "
    3     0.00000000        <        "adventure   "
    4     0.11000000        =        "adventure   "
    5     0.00000000        <       "business    "
    6     0.11040000        =       "business    "
    7     0.00000000        <       "computer    "
    8     0.11640000        =       "computer    "
    9     0.00000000        <       "cooking     "
   10     0.11080000        =       "cooking     "
   11     0.00000000        <       "news        "
   12     0.10660000        =       "news        "
   13     0.00000000        <       "psychology  "
   14     0.11180000        =       "psychology  "
   15     0.00000000        <       "romance     "
   16     0.10800000        =       "romance     "
   17     0.00000000        <       "travel      "
   18     0.11100000        =       "travel      "
```

For example, 10.66% of the values in the type column are “news,” so for a table with 5000 rows, the optimizer estimates that 533 rows will be returned.

Histograms for columns with sparse and dense values

For tables with some values that are highly duplicated, and others that have distributed values, the histogram output shows a combination of operators and a mix of frequency cells and range cells.

The column represented in the histogram below has a value of 30.0 for a large percentage of rows, a value of 50.0 for a large percentage of rows, and a value 100.0 for another large percentage of rows.

There are two steps in the histogram for each of these values: one step representing the highly duplicated value has the = operator and a weight showing the percentage of columns that match the value. The other step for each highly duplicated value has the < operator and a weight of 0.0. The datatype for this column is numeric(5,1).

```
Step      Weight               Value
    1     0.00000000     <=     0.9
    2     0.04456094     <=     20.0
    3     0.00000000     <      30.0
    4     0.29488859     =      30.0
    5     0.05996068     <=     37.0
    6     0.04292267     <=     49.0
    7     0.00000000     <      50.0
    8     0.19659241     =      50.0
    9     0.06028834     <=     75.0
   10     0.05570118     <=     95.0
   11     0.01572739     <=     99.0
   12     0.00000000     <     100.0
```

```
      13     0.22935779      =     100.0
```

Since the lowest value in the column is 1.0, the step for the null values is represented by 0.9.

Choosing the number of steps for highly duplicated values

The histogram examples for frequency cells in this section use a relatively small number of highly duplicated values, so the resulting histograms require less than 20 steps, which is the default number of steps for create index or update statistics.

If your table contains a large number of highly duplicated values for a column, and the distribution of keys in the column is not uniform, increasing the number of steps in the histogram allows the optimizer to produce more accurate cost estimates for queries with search arguments on the column.

For columns with dense frequency counts, the number of steps should be at least one greater than the number of values, to allow a step for the cell representing null values.

For columns with sparse frequency counts, use at least twice as many steps as there are distinct values. This allows for the intervening cells with zero weights, plus the cell to represent the null value. For example, if the titles table in the pubtune database has 30 distinct prices, this update statistics command creates a histogram with 60 steps:

```
update statistics titles
using 60 values
```

This create index command specifies 60 steps:

```
create index price_ix on titles(price)
with statistics using 60 values
```

If a column contains some values that match very few rows, these may still be represented as range cells, and the resulting number of histogram steps will be smaller than the requested number. For example, requesting 100 steps for a state column may generate some range cells for those states represented by a small percentage of the number of rows.

Configuring the histogram tuning factor

histogram tuning factor controls the number of steps Adaptive Server analyzes per histogram for update statistics, update index statistics, update all statistics, and create index.

histogram tuning factor minimizes the resources histograms consume, and only increases resource usage only when it improves optimization. For example, when columns have duplicated values, or data that is not distributed uniformly. Under these circumstances, up to 400 (maximum) histogram steps are used. However, in most cases, Adaptive Server uses the default value (20 in the example above).

For Adaptive Server versions earlier than 15.0.2 ESD #2, the default value for this configuration parameter is 1 (disabled). For Adaptive Server versions 15.0.2 ESD #2 and later, the default for this configuration parameter is 20 (enabled).

See histogram tuning factor in Chapter 5, “Setting Configuration Parameters” in the *System Administration Guide: Volume 1*.

How many histogram steps?

The default value for number of histogram steps (20) may be adequate for columns with a small number of distinct values (that is, “low cardinality”), or for tables with a small number of rows. However, for tables with a large number of rows and that have columns with many distinct values, the default value may be insufficient, particularly when values for the column are not evenly distributed over the rows.

If you think the query processor is generating suboptimal query plans, try increasing the granularity of histograms by increasing their number of steps. Increasing their number of histogram steps can lead to higher resource consumption (particularly procedure cache usage) and longer optimization times before the query processor generates a query plan.

The optimization timeout limit configuration parameter may cause the query processor to generate suboptimal query plans when you increase the number of histogram steps.

You must determine whether a higher number of histogram steps results in better query plans and better overall performance. You may want to increase the value of number of histogram steps to 200; if this does not improve the query plans and performance, try a higher number, like 500.

Alternatively, use one histogram step for every 10,000 data pages. However, in general, you are unlikely to see any improvement by increasing the number of steps to more than 1000 – 2000. If you do not see an improvement in the query plans or performance after changing the number of histogram steps, you should look elsewhere for the bottleneck.

Adaptive Server determines the number of histogram steps when you execute update index statistics:

1. number of histogram steps defines the default for all create index and update index statistics commands for new histograms. The default value for number of histogram steps is 20.
2. You can explicitly set the number of histogram steps to the *nnn* value for update index statistics using:

   ```
   update index statistics table_name
   using nnn values
   ```

3. update index statistics uses the current number of steps in the existing histogram when it creates a new histogram. The number of histogram steps configuration parameter does not apply to existing histograms. update [index] statistics overwrites an existing histogram only when you explicitly specify the number of steps with `using` ***nnn*** `values`.

   When this step completes, the value for the number of histogram steps configuration parameter is the target number of steps for the histogram. This is displayed in optdiag output as “Requested step count.”

4. Adaptive Server multiplies the value for *nnn* (determined in step 2) by the value for histogram tuning factor to generate an internal, intermediate histogram. For example, if *nnn* is 100 and histogram tuning factor is 20, the intermediate histogram may have up to 2000 steps (20 * 100 = 2000). Generating this internal histogram allows Adaptive Server to increase the chance of identifying “frequency cells,” which include duplicate data values. If Adaptive Server finds no frequency cells, it reverts the histogram to the original number of steps. Adaptive Server keeps any frequency cells it finds.

   The original number of steps are displayed in optdiag output as “Requested step count.” optdiag displays the actual number of histogram steps in its output as “Actual step count.”

5 In versions earlier than 15.0.1 ESD #1, Adaptive Server set histogram tuning factor to 1 by default. For version 15.0.1 ESD#1 and later, Adaptive Server uses a default value of 20 for histogram tuning factor. Sybase® recommends that you use the default of 20 unless directed otherwise by Sybase Technical Support.

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. However, 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 re-running update statistics.

Do not attempt to change any statistics by running an update, delete, or insert command.

You can use optdiag output from a 32-bit Adaptive Server to change statistics in another 32-bit Adaptive Server, but not a 64-bit Adaptive Server. Similarly, do not use optdiag output from a 64-bit Adaptive Server as input to a 32-bit Adaptive Server.

After you change statistics using optdiag, running create index or update statistics overwrites the changes. The commands succeed, but print a warning message:

```
WARNING: Edited statistics are overwritten. Table: ’titles’
(objectid 208003772), column: ’type’.
```

Using *optdiag binary*

Note Output displays in hexadecimal not in binary.

Because precision can be lost with floating point numbers, optdiag provides a binary option. The following command displays both human-readable and binary statistics:

```
optdiag binary statistics pubtune..titles.price
        -Usa -Ppasswd -o price.opt
```

In binary, any statistics that can be edited with optdiag are printed twice, once with binary values, and once with floating-point values. The lines displaying the float values start with the optdiag comment character, the pound sign (#).

This sample shows the first few rows of the histogram for the city column in the authors table:

```
Step  Weight              Value

  1   0x3d2810ce    <=    0x41504f204d69616d68ffffffffffffffffffff
# 1   0.04103165    <=    "APO Miami\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 floating-point values instead of the binary values, remove the pound sign from the lines displaying the floating-point values, and insert the pound sign at the beginning of the corresponding line displaying the binary value.

When to use *binary*

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 differ. In cases such as this, you should use binary for input.

If you do not use binary mode, optdiag issues an error message indicating that the step values are not increasing and advising you to use binary mode. To avoid losing precision in sysstatistics, optdiag skips loading the histogram in which the error occurred.

Updating selectivities with *optdiag* input mode

The optimizer uses range and in-between selectivity values when the value of a search argument is unknown during queryoptimization. You can use optdiag to customize the server-wide default selectivity values to match the data for specific columns in your application.

The server-wide defaults are:

- Range selectivity – 0.33
- In-between selectivity – 0.25

The following example shows how optdiag displays default values:

```
Statistics for column:              "city"
Last update of column statistics:   Feb  4 2008  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 the selectivity values, replace the entire “default used (0.33)” or “default used (0.25)” string with a float value. The following example changes the range selectivity to .25 and the in-between selectivity to .05, using decimal values:

```
     Range selectivity:            0.250000000
     In between selectivity:        0.050000000
```

Editing histograms

You can edit histograms to:

- Remove a step, by transferring its weight to an adjacent line and deleting the step
- Add a step or steps, by spreading the weight of a cell to additional lines, with the upper bound for column values any new step is to represent

Adding frequency count cells to a histogram

One reason for editing histograms is to add frequency count cells without greatly increasing the number of steps.

Editing a histogram with a dense frequency count

The changes you make to histograms vary, depending on whether the values represent a dense or sparse frequency count. To add a frequency cell for a given column value, check the column value just less than the value for the new cell. If the next-lesser value is as close as possible to the value to be added, then you can determine the frequency count.

If the next-lesser column value changed is as close as possible to the frequency count value, you can easily extract the frequency count cell.

For example, if a column contains at least one 19 and many 20s, and the histogram uses a single cell to represent all the values greater than 17 and less than or equal to 22, optdiag output shows the following information for the cell:

```
Step      Weight                Value
...
4         0.100000000    <=     17
5         0.400000000    <=     22
...
```

Altering this histogram to place the value 20 on its own step requires adding two steps, as shown here:

```
...
4     0.100000000    <=     17
5     0.050000000    <=     19
6     0.300000000    <=     20
7     0.050000000    <=     22
...
```

In the altered histogram above, step 5 represents all values greater than 17 and less than or equal to 19. The sum of the weights of steps 5, 6, and 7 in the modified histogram equals the original weight value for step 5.

Editing a histogram with a sparse frequency count

If the column has no values greater than 17 and less than 20, use the representation for a sparse frequency count. 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 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. Violated rules generate error messages.

- Each step number should be one more than the previous step number (unless -T4 is on). For column values for each step, the value should be greater than or equal to the previous step's column value.
- 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

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, you can use simulated statistics to:

- Help Technical Support replicate optimizer problems
- Perform “what-if” analyses to plan configuration changes
- Perform diagnostics on a development server.

See “Requirements for loading and using simulated statistics” on page 47.

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 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

To display simulate-mode statistics for the pubtune database, use:

```
optdiag simulate statistics pubtune -o pubtune.sim
```

To generate 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 you specify binary mode, there are three rows of output:
    - A binary "simulated" row
    - A decimal "simulated" row, commented out
    - A decimal "actual" row, commented out
- If you do not specify binary mode, there are two rows:
    - A "simulated" row
    - An "actual" row, commented out

Here is a sample of the table-level statistics for the authors table in the pubtune database:

```
Table owner:                                   "dbo"
Table name:                                      "authors"

Statistics for table:                            "authors"

    Partition count:                             3

Statistics for partition:                        "authors_1376004902"
    Data page count:                             74
    Empty data page count:                       0
    Data row count:                              1666.0000000000000000
    Forwarded row count:                         0.0000000000000000
    Deleted row count:                           0.0000000000000000
    Data page CR count:                          10.0000000000000000
    OAM + allocation page count:                 3
    First extent data pages:                     0
    Data row size:                               85.2623049219687914
    Parallel join degree:                        0.0000000000000000
    Unused page count:                    5
OAM page count:                           1

Derived statistics:
    Data page cluster ratio:              1.0000000000000000
    Space utilization:                    0.9521597490347491
    Large I/O efficiency:                 1.0000000000000000
```

In addition to table and index statistics, simulate optdiag output includes:

- Partitioning information for partitioned tables. If a table is partitioned, the simulated and actual information is printed for each partition of the table. The “Pages in the largest partition” line is not relevant:

```
     Pages in largest partition:      390.0000000000000000 (simulated)
#    Pages in largest partition:       390.0000000000000000 (actual)
```

- Settings for the parallel processing configuration parameters:

```
Configuration Parameters:
     Number of worker processes:      20 (simulated)
#    Number of worker processes:      20 (actual)
     Max parallel degree:             10 (simulated)
#    Max parallel degree:             10 (actual)
     Max scan parallel degree:        3 (simulated)
#    Max scan parallel degree:         3 (actual)
```

- Cache configuration information for the default data cache and the caches used by the specified database or the specified table and its indexes. If tempdb is bound to a cache, that cache’s configuration is also included. Here is sample output for the cache used by the pubtune database:

```
Configuration for cache:             "pubtune_cache"

     Size of 2K pool in Kb:          15360 (simulated)
#    Size of 2K pool in Kb:          15360 (actual)
     Size of 4K pool in Kb:          0 (simulated)
#    Size of 4K pool in Kb:          0 (actual)
     Size of 8K pool in Kb:          0 (simulated)
#    Size of 8K pool in Kb:          0 (actual)
     Size of 16K pool in Kb:         0 (simulated)
#    Size of 16K pool in Kb:          0 (actual)
```

To test how queries use a 16K pool, alter the simulated statistics values above to read:

```
Configuration for cache:             "pubtune_cache"

     Size of 2K pool in Kb:           10240 (simulated)
```

```
#    Size of 2K pool in Kb:            15360 (actual)
     Size of 4K pool in Kb:            0 (simulated)
#    Size of 4K pool in Kb:            0 (actual)
     Size of 8K pool in Kb:            0 (simulated)
#    Size of 8K pool in Kb:            0 (actual)
     Size of 16K pool in Kb:           5120 (simulated)
#    Size of 16K pool in Kb:            0 (actual)
```

Requirements for loading and using simulated statistics

To use simulated statistics, issue set statistics simulate on before running the query.

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 nondefault cache strategies and any nondefault concurrency optimization values
- Bind databases and objects to the caches used in the environment you are simulating
- Include any set options that affect query optimization (such as set parallel_degree) in the batch you are testing
- Create any view used in the query
- Use cursors, if they are used for the query
- Use a stored procedure, if you are simulating a procedure execution

You can load simulated statistics into the original database, or into a database created solely for performing “what-if” analyses on queries.

Using simulated statistics in the original database

Statistics are loaded into the original database in separate rows in the system tables; they do not overwrite existing nonsimulated statistics. Simulated statistics are used only 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 suboptimal for the actual tables and indexes, and executing these queries may adversely affect other queries on the system.

Using simulated statistics in another database

When statistics are loaded into a database created solely for performing "what-if" analyses on queries, you must first veriy that:

- The database named in the input file exists; 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 exists. The tables do not need to contain data.
- All caches named in the input file exist. They can be the smallest possible cache size, 512K, with only a 2K pool. The simulated statistics provide the information for pool configuration.

Dropping simulated statistics

Loading simulated statistics adds rows that describe cache configuration to the sysstatistics table in the master database. To remove these statistics, use delete shared statistics. The command has no effect on the statistics in the database where the simulated statistics were loaded.

If you have loaded simulated statistics into a database that contains real table and index statistics, you can drop simulated statistics by either:

- Using delete statistics on the table, which deletes all statistics, and running update statistics to re-create only the nonsimulated statistics, or,
- Using optdiag (without simulate mode) to copy statistics out; then running delete statistics on the table, and using optdiag (without simulate mode) to copy statistics in.

Running queries with simulated statistics

Use set statistics simulate on to optimize queries using simulated statistics:

```
set statistics simulate on
```

In most cases, you also want to use set showplan on.

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 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:

```
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 displays an adjusted query plan. If set noexec is turned on, the adjusted plan does not appear.

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. For example, 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 (#), which, in input mode, means that all characters on the line following a pound sign are ignored, except when the pound sign occurs within quotation marks as part of a column name or column value.

Effects of SQL commands on statistics

The information stored in systabstats and sysstatistics is affected by data definition language (DDL). Some data modification language (DML) also affects systabstats. Table 2-8 summarizes how DDL affects the systabstats and sysstatistics tables.

Table 2-8: Effects of DDL on systabstats and sysstatistics

Command	Effect on systabstats	Effect on sysstatistics
alter table...lock	Changes values to reflect the changes to table and index structure and size. When you change tables and indexes 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 you are changing from allpages to data-only locking or back, no effect on changing between data-only locking schemes.
alter table to add, drop or modify a column definition	Some alter table parameters (for example, using drop column, adding non-NULL columns with add, or using modify to decrease the length of variable-length columns) require a data copy of the table. Other alter table operations are done with an update of the system catalog information. If the change affects the length of the row and alter table must copy the table, the alter table parameter changes the values to reflect the changes to table and index structures and size. If the alter table parameter does not perform a data copy, then no changes are done.	If the change requires a data copy, alter table rebuilds all indexes, and has the same effect as create clustered or create non-clustered index. alter table does not affect the statistics row maintained on nonindexed columns, but it does delete statistics rows on columns being dropped. If the change does not perform a data copy, no changes are done.
create table	Adds a row for the table. If a constraint creates an index, see the create index commands below.	Adds a row for the table. If 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.

Command	Effect on systabstats	Effect on sysstatistics
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 indexed columns. This allows the optimizer to continue to use this information. Deletes simulated statistics for nonclustered indexes. For clustered indexes on allpages-locked tables, changes the value for the index ID in the row that contains simulated table data.
drop table	Removes all rows for the table.	Removes all rows for the table.
reorg	Updates restart points, if used with a time limit; updates number of pages and cluster ratios if page counts change; affects other values such as empty pages, forwarded, or deleted row counts, depending on the option used.	The rebuild option re-creates indexes.
truncate table	Resets values to reflect an empty table. Some values, like row length, are retained.	No effect; this allows reloading a truncated table without re-running update statistics.
update statistics (Running update statistics on an empty table does not affect the system tables.)		
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.

Command	Effect on systabstats	Effect on sysstatistics
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 densities for all indexes and prefix subsets of indexes.

How query processing affects *systabstats*

Data modification can affect many of the values in the systabstats table. To improve performance, these values are changed in memory and flushed to systabstats periodically by the housekeeper chores task.

To query systabstats directly, use sp_flushstats to flush the in-memory statistics to systabstats. For example, to flush the statistics for the titles table and any indexes on the table, use:

```
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 and histograms using *sp_showoptstats*

Adaptive Server includes sp_showoptstats, which functions like the optdiag standalone utility, extracting and showing, in an XML document, statistics and histograms for various types of data objects from system tables such as systabstats and sysstatistics. The syntax is:

sp_showoptstats [*dbname*[.*owner*[.[*table_name*]]]], [.*column*],
    [*option*]

See sp_showoptstats in *Adaptive Server Reference Manual: Procedures*.

Index