



PERFORMANCE AND TUNING GUIDE | PUBLIC
SAP Adaptive Server Enterprise 16.0 SP03
Document Version: 1.0 – 2020-03-04

Performance and Tuning Series: Physical Database Tuning

Content

- 1 Control Physical Data Placement. 7**
- 1.1 Improve Performance by Controlling Object Placement. 7
 - Identify Poor Object Placement. 8
 - Use sp_sysmon While Changing Data Placement. 8
 - Page Validation Signatures. 9
- 1.2 Improve I/O Performance. 9
 - Spread Data Across Disks to Avoid I/O Contention. 10
 - Isolate Server-Wide I/O from Database I/O. 11
 - Keep Transaction Logs on a Separate Disk. 11
 - Mirror a Device on a Separate disk. 12
- 1.3 Segment Usage. 13
 - Create Objects on Segments. 13
 - Separate Tables and Indexes. 14
 - Split Large Tables Across Devices. 14
 - Move Text Storage to a Separate Device. 15
- 1.4 Partition Tables for Performance. 15
 - How SAP ASE Distributes Partitions on Devices. 16
- 1.5 Space Planning for Partitioned Tables. 16
 - Read-Only Tables. 17
 - Read-Mostly Tables. 17
 - Tables with Random Data Modification. 18
- 1.6 Disks and Full Devices. 18
 - Add Disks When Devices are Full. 19
 - Add Disks When Devices are Nearly Full. 20
- 1.7 Maintenance Issues and Partitioned Tables. 21
 - Regular Maintenance Checks for Partitioned Tables. 21
- 2 Data Storage. 22**
- 2.1 Query Optimization. 22
 - Query Processing and Page Reads. 22
- 2.2 SAP ASE Pages. 23
 - Page Headers and Page Sizes. 24
 - Data and Index Pages. 24
 - Large Object (LOB) Pages. 25
 - Extents. 25
- 2.3 Pages that Manage Space Allocation. 26
 - Global Allocation Map (GAM) Pages. 26

	Allocation Pages.	27
	Object Allocation Map Pages.	27
	How OAM Pages and Allocation Pages Manage Object Storage.	27
	Page Allocation Keeps an Object's Pages Together.	28
	Data Access Using sysindexes and syspartitions.	29
2.4	Space Overheads.	30
	Number of Columns and Size.	30
	Number of Rows per Data Page.	36
	Additional Number of Object and Size Restrictions.	36
2.5	Tables without Clustered Indexes.	36
	Lock Schemes.	37
	select Operations on Heap Tables.	38
	Data Insertion into an Allpages-Locked Heap Table.	38
	Data Insertion into a Data-Only-Locked Heap Table.	39
	Data Deletion from a Heap Table.	40
	Update Data on a Heap Table.	41
	How SAP ASE Performs I/O for Heap Operations.	42
	Heap Table Maintenance.	42
	Transaction Log: A Special Heap Table.	44
	Asynchronous Prefetch and I/O on Heap Tables.	44
2.6	Caches and Object Bindings.	45
	Heap Tables, I/O, and Cache Strategies.	45
	select Operations and Caching.	47
	Data Modification and Caching.	48
3	Setting Space Management Properties.	50
3.1	Index Maintenance Reduction.	50
	Advantages of Using fillfactor.	51
	Disadvantages of Using fillfactor.	51
	Set fillfactor Values.	51
	fillfactor Examples.	52
	sorted_data and fillfactor Option Usage.	55
3.2	Row Forwarding Reduction.	56
	Default, Minimum, and Maximum Values for exp_row_size.	57
	Specify an Expected Row Size with Create Table.	57
	Add or Change an Expected Row Size.	58
	Set a Default Expected Row Size Server-Wide.	58
	Display the Expected Row Size for a Table.	59
	Choose an Expected Row Size for a Table.	59
	Conversion of max_rows_per_page to exp_row_size.	60
	Monitoring and Managing Tables That use Expected Row Size.	61
3.3	Space for Forwarded Rows and Inserts.	61

	Extent Allocation Commands and reservepagegap.	62
	Specify a Reserve Page Gap with create table.	63
	Specify a Reserve Page Gap with create index.	63
	Change reservepagegap.	63
	reservepagegap Examples.	64
	Choose a Value for reservepagegap.	65
	Monitor reservepagegap Settings.	66
	reservepagegap and sorted_data Options.	66
3.4	max_rows_per_page Usage on allpages-locked Tables.	68
	Lock Contention Reduction.	69
	Indexes and max_rows_per_page.	69
	select into and max_rows_per_page.	70
	Applying max_rows_per_page to Existing Data.	70
4	Table and Index Size.	71
4.1	Determine the Sizes of Tables and Indexes.	71
4.2	Effects of Data Modifications on Object Sizes.	72
4.3	optdiag Usage to Display Object Sizes.	72
	Advantages of optdiag.	72
	Disadvantages of optdiag.	73
4.4	sp_spaceused Usage to Display Object Size.	73
	Advantages of sp_spaceused.	74
	Disadvantages of sp_spaceused.	74
4.5	Using sp_estspace to Estimate Object Size.	75
	Advantages of sp_estspace.	76
	Disadvantages of sp_estspace.	76
4.6	Use Formulas to Estimate Object Size.	76
	Factors that can Affect Storage Size.	76
	Storage Sizes for datatypes.	77
	Tables and Indexes Used in the Formulas.	78
	Calculating Table and Clustered Index Sizes for allpages-locked Tables.	79
	Calculating the Sizes of Data-Only-Locked Tables.	88
	Other Factors Affecting Object Size.	94
	Very Small Rows.	96
	LOB Pages.	96
	Advantages of Using Formulas to Estimate Object Size.	97
	Disadvantages of Using Formulas to Estimate Object Size.	97
5	Database Maintenance.	99
5.1	Run reorg on Tables and Indexes.	99
5.2	Create and Maintain Indexes.	100
	Configure SAP ASE to Speed Sorting.	100

	Dump the Database After Creating an Index.	100
	Create an Index on Sorted Data.	100
	Maintain Index and Column Statistics.	101
	Rebuild Indexes.	102
5.3	Create or Alter a Database.	103
5.4	Backup and Recovery.	104
	Local Backups.	104
	Remote Backups.	104
	Online Backups.	104
	Use Thresholds to Prevent Running Out of Log Space.	105
	Minimize Recovery Time.	105
	Recovery Order.	105
5.5	Bulk-Copy.	105
	Parallel Bulk-Copy.	106
	Batches and Bulk-Copy.	106
	Slow Bulk-Copy.	106
	Improve Bulk-Copy Performance.	107
	Replacing the Data in a Large Table.	107
	Add Large Amounts of Data to a Table.	107
	Use Partitions and Multiple Bulk-Copy Processes.	108
	Impact on Other Users.	108
5.6	Database Consistency Checker.	108
5.7	Use dbcc tune (cleanup).	108
5.8	Use dbcc tune on Spinlocks.	109
5.9	Determine the Space Available for Maintenance Activities.	110
	Overview of Space Requirements.	110
	Check Space Usage and Available Space.	110
	Estimate the Effects of Space Management Properties.	113
	If There is Not Enough Space.	114
6	Temporary Databases.	115
6.1	How Temporary Database Management Affects Performance.	115
6.2	Use Temporary Tables.	116
	Hashed (#) Temporary Tables.	116
	Regular User Tables.	116
	Worktables.	117
6.3	Temporary Databases.	117
6.4	Session-Assigned Temporary Database.	117
6.5	Create User Temporary Databases.	118
6.6	Configure a Default tempdb Group.	118
6.7	Bind to Groups and tempdb.	119
	Bind applications and Logins to Temporary Databases.	119

6.8	Tune System Temporary Databases for Performance.	120
	Placing System tempdb.	120
	Configure User-Created Temporary Databases.	122
	General Guidelines.	123
6.9	Log Optimizations for Temporary Databases.	128
	User Log Cache (ULC).	128

1 Control Physical Data Placement

Improve performance by controlling the location of tables and indexes.

To make the most of physical database tuning, understand these distinctions between logical and physical devices:

- The physical disk or physical device is the hardware that stores the data.
- A database device or logical device is all or part of a physical disk that has been initialized (with the `disk init` command) for use by SAP Adaptive Server Enterprise (SAP ASE). A database device can be an operating system file, an entire disk, or a disk partition.
See the *Installation Guide* and the *Configuration Guide* for your platform for information about specific operating system constraints on disk and file usage.
- A segment is a named collection of database devices used by a database. The database devices that make up a segment can be located on separate physical devices.
- A partition is a subset of a table. Partitions are database objects that can be managed independently. You can split partitioned tables, so multiple tasks can access it simultaneously. You can place a partition on a specific segment. If each partition is on a different segment and each segment has its own database device, queries accessing these tables benefit from improved parallelism. See `create table` in the *Reference Manual: Commands* and the *Transact-SQL Users Guide* for more information about creating and using partitions.
- To use parallel `bcp` on a UNIX platform use `bcp_r`, rather than `bcp`.

Use `sp_helpdevice`, `sp_helpsegment`, and `sp_helppartition` to get more information about devices, segments, and partitions.

1.1 Improve Performance by Controlling Object Placement

SAP ASE allows you to control the placement of databases, tables, and indexes across physical storage devices, which can improve performance by equalizing the reads and writes to disk across many devices and controllers.

For example, you can:

- Place database data segments on a specific device or devices, storing the database log on a separate physical device so that reads and writes to the log do not interfere with data access.
- Spread large, heavily used tables across several devices.
- Place specific tables or nonclustered indexes on specific devices. For example, you might place a table on a segment that spans several devices and its nonclustered indexes on a separate segment.
- Place the `text` and `image` page chain for a table on a separate device from the table. The table stores a pointer to the actual data value in the separate database structure, so each access to a `text` or `image` column requires at least two I/Os.
- Distribute tables evenly across partitions on separate physical disks to provide optimum parallel query performance and improve `insert` and `update` performance.

For multiuser and multi-CPU systems that perform a lot of disk I/O, be especially aware of physical and logical device issues and the distribution of I/O across devices:

- Plan a balanced separation of objects across logical and physical devices.
- Use enough physical devices, including disk controllers, to ensure physical bandwidth.
- Use an increased number of logical devices to ensure minimal contention for internal I/O queues.
- Determine and use a number of partitions that allows parallel scans and meets query performance goals.

1.1.1 Identify Poor Object Placement

Object placement can effect your system's performance.

Your system may benefit from more appropriately placed objects if:

- Single-user performance is satisfactory, but response time increases significantly when SAP ASE executes multiple processes.
- Access to a mirrored disk takes twice as long as access to an unmirrored disk.
- Objects that are frequently accessed ("hot objects") degrade the performance of queries that use the tables in which these objects are located.
- Maintenance activities take a long time.
- `tempdb` performance is affected if it shares disk space with other databases. Most system procedures and applications use `tempdb` as their workspace, and are adversely affected if `tempdb` shares the same disk with other databases.
- `insert` performance is poor on heavily used tables.
- Queries that run in parallel perform poorly, due to an imbalance of data pages on partitions or devices, or they run in serial, due to extreme imbalance.

If you experience problems due to disk contention and other problems related to object placement, check for and correct these issues:

- Random-access (I/O for data and indexes) and serial-access (log I/O) processes use the same disks.
- Database processes and operating system processes use the same disks.
- Serial disk mirroring.
- Database logging or auditing takes place on the same disk as data storage.

1.1.2 Use `sp_sysmon` While Changing Data Placement

Use `sp_sysmon` to determine whether data placement across physical devices is causing performance problems. Check the entire `sp_sysmon` output during tuning to verify how the changes affect all performance categories.

Pay special attention to the output associated with:

- I/O device contentions
- All-pages locked heap tables
- Last page locks on heaps
- Disk I/O management

See *Monitoring SAP ASE with sp_sysmon*.

1.1.3 Page Validation Signatures

SAP ASE includes page validation signatures to data pages.

Page validation signatures mark the data page, allowing SAP ASE to detect whether a page that it read from disk has changed since it was written to disk. Page validation signatures provide a means of detecting I/O problems (such as a partial write or on-disk corruption).

However, marking the data pages also changes the page, and signed databases are not backwardly compatible to SAP ASE versions that do not have signed pages. A dump of a signed database cannot be loaded onto a server that doesn't have the feature. You can clear these signatures for the purpose of downgrade or dump compatibility.

Use `sp_dboption 'allow page signing'` to enable and disable page signing by setting the parameter to true and false, respectively. However, disabling page signing does not undo the signatures of pages that were signed while the option was enabled. Use the `sign_pages` function to undo existing page signatures (see *New and Changed Functions*).

Use the `sign_pages` function to mark pages as signed or unsigned, and determine if a database is ready for downgrade.

i Note

The Cluster Edition does not support page validation signatures. Enabling `sp_dboption 'allow page signing'` in the Cluster Edition does not result in a warning or error message, but pages are not signed.

1.2 Improve I/O Performance

Suggestions for improving I/O performance.

Try:

- Spreading data across disks to avoid I/O contention
- Isolating server-wide I/O from database I/O
- Separating data storage and log storage for frequently updated databases
- Keeping random disk I/O away from sequential disk I/O
- Mirroring devices on separate physical disks
- Using partitions to distribute table data across devices

1.2.1 Spread Data Across Disks to Avoid I/O Contention

Avoid bottlenecks by spreading data storage across multiple disks and multiple disk controllers.

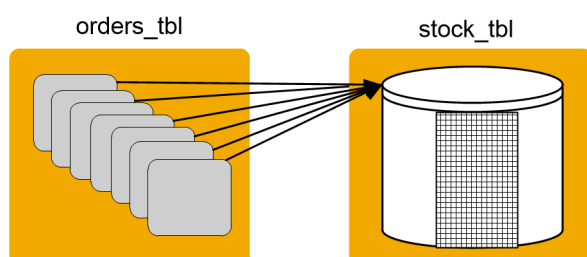
- Place databases with critical performance requirements on separate devices. If possible, also use separate controllers than those used by other databases. Use segments as needed for critical tables, and partitions as needed for parallel queries.
- Put heavily used and frequently joined tables on separate disks.
- Use segments to place tables and indexes on their own disks.

1.2.1.1 Avoid Physical Contention in Parallel Join Queries

An example illustrating a join of two tables, `orders_tbl` and `stock_tbl`.

There are 10 worker process available: `orders_tbl` has 10 partitions on 10 different physical devices and is the outer table in the join; `stock_tbl` is nonpartitioned. The worker processes have a problem with access contention on `orders_tbl`, but each worker process must scan `stock_tbl`. There may be physical I/O contention if the entire table does not fit into cache. In the worst case, 10 worker processes attempt to access the physical device on which `stock_tbl` resides. Avoid physical I/O contention by creating a named cache that contains the entire table `stock_tbl`.

Another way to reduce or eliminate physical I/O contention is to partition both `orders_tbl` and `stock_tbl` and distribute those partitions on different physical devices.



1.2.2 Isolate Server-Wide I/O from Database I/O

Place system databases with heavy I/O requirements (for example, `tempdb` and `sybsecurity`) on physical disks and controllers other than where application databases reside.

1.2.2.1 `tempdb`

It is a heavily used database that affects all processes on the server and is used by most system procedures. It is automatically installed on the master device. If more space is needed, you can expand `tempdb` to other devices.

If you expect `tempdb` to be quite active, place it on a disk—the fastest available—that is not used for other important database activity.

On some UNIX systems, I/O to operating system files is significantly faster than I/O to raw devices. `tempdb` is always re-created, rather than recovered, after a shutdown; therefore, you may be able to improve performance by moving `tempdb` onto an operating system file instead of a raw device. Test this on your own system.

Related Information

[Temporary Databases \[page 115\]](#)

1.2.2.2 `sybsecurity`

Once enabled, the auditing system performs frequent I/O to the `sysaudits` table in the `sybsecurity` database. If your applications perform a significant amount of auditing, place `sybsecurity` on a disk that is used for tables where fast response time is not critical. Ideally, place `sybsecurity` on its own device.

Use the threshold manager to monitor free space to avoid suspending user transactions if the audit database fills up. See Chapter 16, “Managing Free Space with Thresholds,” in *System Administration Guide, Volume 2* for information about determining appropriate thresholds.

1.2.3 Keep Transaction Logs on a Separate Disk

Place transaction logs on a separate segment, preventing the logs from competing with other objects for disk space.

Placing the logs on a separate physical disk:

- Improves performance by reducing I/O contention
- Ensures full recovery in the event of hard disk failures on the data device

- Speeds recovery, since simultaneous asynchronous prefetch requests can read ahead on both the log device and the data device without contention

Both `create database` and `alter database` require you to use `with override` before you can place the transaction log on the same device as the data.

The log device can experience significant I/O on systems with heavy update activity. SAP ASE writes log pages to disk when transactions commit, and may need to read log pages into memory to replace deferred updates with deferred operations.

When log and data are on the same database devices, the extents allocated to store log pages are not contiguous; log extents and data extents are mixed. When the log is on its own device, SAP ASE allocates the extents sequentially, thus reducing disk head travel and seeks, and maintaining a higher I/O rate.

SAP ASE buffers log records for each user in a user log cache, which reduces contention for writing to the log page in memory. If log and data are on the same devices, user log cache buffering is disabled, which results in serious performance degradation on SMP systems.

See Chapter 6, “Overview of Disk Resource Issues,” in the *System Administration Guide: Volume 1*.

1.2.4 Mirror a Device on a Separate disk

Disk mirroring is a high availability feature that allows SAP ASE to duplicate the contents of an entire database device.

See Chapter 2, “Disk Mirroring,” in the *System Administration Guide, Volume 2*.

If you mirror data, put the mirror on a separate physical disk from the device that it mirrors, minimizing mirroring’s performance impact. Disk hardware failure often results in whole physical disks being lost or unavailable

If you do not use mirroring, or use operating system mirroring, you may see slight performance improvements by setting `disable disk mirroring` configuration parameter to 1.

Mirroring can increase the time taken to complete disk writes, since the writes are executed on both disks, either serially or simultaneously. Disk mirroring has no effect on the time required to read data.

Mirrored devices use one of two modes for disk writes:

- Nonserial mode – can require more time to complete a write than an unmirrored write requires. In nonserial mode, both writes start at the same time, and SAP ASE waits for both to complete. The time to complete nonserial writes is the greater of the two I/O times.
- Serial mode – increases the time required to write data even more than nonserial mode. SAP ASE starts the first write and waits for it to complete before starting the second write. The time required is the sum of the two I/O times.

1.2.4.1 Serial Mode Usage

Despite its performance impact, serial mode is the default mode because it guards against failures that occur while a write is taking place.

Since serial mode waits until the first write is complete before starting the second write, a single failure cannot affect both disks. Using nonserial mode improves performance, but you risk losing data if a failure occurs that affects both writes.

⚠ Caution

If your mirrored database system must be absolutely reliable, use serial mode.

1.3 Segment Usage

A segment is a label that points to one or more logical devices.

Use segments to improve throughput by:

- Splitting large tables across disks, including tables that are partitioned for parallel query performance
- Separating tables and their nonclustered indexes across disks
- Separating table partitions and index across the disks
- Placing the text and image page chain on a disk other than the one on which the table resides, where the pointers to the text values are stored

In addition, you can use segments to control space usage:

- Tables or partitions cannot grow larger than their segment allocation. You can use segments to limit the table or partition size.
- Tables or partitions on other segments cannot use the space allocated to objects on another segment.
- The threshold manager monitors space usage.

1.3.1 Create Objects on Segments

Each database can use up to 32 segments, including the 3 segments that are created by the system (`system`, `log segment`, and `default`) when a database is created.

Tables and indexes are stored on segments. If you execute `create table` or `create index` without specifying a segment, the objects are stored on the `default` segment for the database. Naming a segment in either of these commands creates the object on that segment. You can use the `sp_placeobject` system procedure to assign all future space allocations to take place on a specified segment, so tables can span multiple segments.

A system administrator must initialize the device with `disk init` and allocate the device to the database. Alternatively, the database owner can do this using `create database` or `alter database`.

Once the devices are available to the database, the database owner or object owners can create segments and place objects on the devices.

When you create a user-defined segment, you can place tables, indexes, and partitions on that segment using the `create table` or `create index` commands:

```
create table tableA(...) on seg1
```

```
create nonclustered index myix on tableB(...)
    on seg2
```

This example creates the table `fictionales`, which is partitioned by range according to values in the `date` column:

```
create table fictionales
(store_id int not null,
order_num int not null,
date datetime not null)
partition by range (date)
(q1 values <= ("3/31/2013") on seg1,
q2 values <= ("6/30/2013") on seg2,
q3 values <= ("9/30/2013") on seg3,
q4 values <= ("12/31/2013") on seg4)
```

By controlling the location of critical tables, you can arrange for these tables and indexes to be spread across disks.

1.3.2 Separate Tables and Indexes

Use segments to place tables on one set of disks and nonclustered indexes on another set of disks.

You cannot place a clustered index on a different segment than its data pages. When you create a clustered index using the `on <segment_name>` clause, the entire table is moved to the specified segment, and the clustered index tree is built on that segment.

You can improve performance by placing nonclustered indexes on a separate segment.

1.3.3 Split Large Tables Across Devices

Segments can span multiple devices, so you can use them to spread data across one or more disks.

This can help balance the I/O load for large and busy tables. For parallel queries, it is essential that you create segments across multiple devices for I/O parallelism during partitioned-based scans.

See Chapter 8, "Creating and Using Segments," in the *System Administration Guide, Volume 2*.

1.3.4 Move Text Storage to a Separate Device

When a table includes a `text`, `image`, or Java off-row datatype, the table itself stores a pointer to the data value. The actual data is stored on a separate linked list of pages called a large object chain (LOB).

Writing or reading a LOB value requires at least two disk accesses, one to read or write the pointer, and one for subsequent reads or writes for the data. If your application frequently reads or writes LOB values, you can improve performance by placing the LOB chain on a separate physical device. Isolate LOB chains on disks that are not busy with other application-related table or index access.

When you create a table with LOB columns, SAP ASE creates a row in `sysindexes` and `syspartitions` for the object that stores the LOB data. The value in the `name` column is the table name prefixed with a "t"; the `indid` is always 255. If you have multiple LOB columns in a single table, there is only one object used to store the data. By default, this object is placed on the same segment as the table.

Use `sp_placeobject` to move all future allocations for the LOB columns to a separate segment.

Reading or writing a separate text or image page can be a slow operation. However, many text or image values are quite short, and can fit in a row instead of a separate page. You can significantly improve the speed of queries that use `text` or `image` columns by specifying that short values are stored in-row.

Use `create table ... in row` and `alter table ... in row` to create a table, or alter an existing table, to stores values in row.

Use `create database ... inrow_lob_length` or `alter database ... inrow_lob_length` to create a database, or alter an existing database, that stores values in row.

See the *Transact-SQL Users Guide > In Row Off Row LOB*.

1.4 Partition Tables for Performance

Partitioning a table can improve performance for several types of processes.

- Partitioning allows parallel query processing to access each partition of the table. Each worker process in a partitioned-based scan reads a separate partition.
- Partitioning allows you to load a table in parallel with bulk copy. For more information on parallel `bcp`, see the *Utility Programs* manual.
- Partitioning allows you to distribute a table's I/O over multiple database devices.
- Semantic partitioning (range-, hash- and list-partitioned tables) improves response time because the query processor eliminates some partitions.
- Partitioning provides multiple insertion points for a heap table.

The tables you choose to partition and the type of partition depend on the performance issues you encounter and the performance goals for the queries on the tables.

See Chapter 10, "Partitioning Tables and Indexes" in the *Transact-SQL Users Guide* book for more information about, and examples using and creating partitions.

1.4.1 How SAP ASE Distributes Partitions on Devices

In SAP ASE 15.0 and later; all partitions are created on the first device when you create partitions on a segment that is mapped to multiple database devices.

In versions earlier than 15.0, SAP ASE automatically maintained an affinity between partitions and devices when you created multiple partitions on a segment that was mapped to multiple database devices. To achieve affinity between partitions and devices:

1. Create a segment for a particular device.
2. Explicitly place a partition on that segment.

You can create as many as 29 user segments, and you must use the `alter table` syntax from SAP ASE version 15.0 and later to create the segments, because the earlier syntax (`alter table t partition 20`) does not support explicit placement of partitions on segments.

Achieve the best I/O performance for parallel queries by matching the number of partitions to the number of devices in the segment.

You can partition tables that use the `text`, `image`, or Java off-row data types. However, the columns themselves are not partitioned—they remain on a single page chain.

1.4.1.1 RAID Devices and Partitioned Tables

A striped redundant array of independent disks (RAID) device can contain multiple physical disks, but SAP ASE treats such a device as a single logical device. You can use multiple partitions on the single logical device and achieve good parallel query performance.

To determine the optimum number of partitions for your application mix, start with one partition for each device in the stripe set. Use your operating system utilities (`vmstat`, `sar`, and `iostat` on UNIX; Performance Monitor on Windows) to check utilization and latency.

To check maximum device throughput, use `select count(*)`, using the `index <table_name>` clause to force a table scan if a nonclustered index exists. This command requires minimal CPU effort and creates very little contention for other resources.

1.5 Space Planning for Partitioned Tables

Considerations for planning partitioned tables.

Consider how to maintain:

- Load balance across the disk for partition-based scan performance and for I/O parallelism
- Cclustered indexes, which require approximately 120% of the space occupied by the table to drop and re-create the index or to run `reorg rebuild`

The space planning decisions you make depend on the:

- Availability of disk resources for storing tables
- Nature of your application mix and of the incoming data (for semantic-partitioned tables)

Estimate the frequency with which your partitioned tables need maintenance: some applications need indexes to be re-created frequently to maintain balance, while others need little maintenance.

For those applications that need frequent load balancing for performance, having space in which to re-create a clustered index or run `reorg rebuild` provides fastest and easiest results. However, since creating clustered indexes requires copying the data pages, the space available on the segment must be equal to approximately 120% of the space occupied by the table.

The following descriptions of read-only, read-mostly, and random data modification provide a general picture of the issues involved in object placement and in maintaining partitioned tables.

See Chapter 10, “Partitioning Tables and Indexes” in the *Transact-SQL Users Guide* for information about the specific tasks required during maintenance.

Related Information

[Determine the Space Available for Maintenance Activities \[page 110\]](#)

1.5.1 Read-Only Tables

Tables that are read-only, or that are rarely changed, can completely fill the space available on a segment, and do not require maintenance. If a table does not require a clustered index, you can use parallel bulk copy (parallel `bcp`) to completely fill the space on the segment.

If a clustered index is needed, the table’s data pages can occupy up to 80% of the space in the segment. The clustered index tree requires about 20% of the space used by the table.

This space requirement varies, depending on the length of the key. Initially, loading the data into the table and creating the clustered index requires several steps, but once you have performed these steps, maintenance is minimal.

1.5.2 Read-Mostly Tables

The guidelines for read-only tables also apply to read-mostly tables with very few inserts.

The only exceptions are:

- If there are inserts to the table, and the clustered index key does not balance new space allocations evenly across the partitions, the disks underlying some partitions may become full, and new extent allocations are made to a different physical disk. This process is called extent stealing. In huge tables spread across many disks, a small percentage of allocations to other devices is not a problem. Detect extent stealing by using `sp_helpsegment` to check for devices that have no space available, and by using `sp_helppartition` to check for partitions that have disproportionate numbers of pages.

If the imbalance in partition size leads to degradation in parallel query response times or optimization, you may want to balance the distribution by using one of the methods described in Chapter 10, “Partitioning Tables and Indexes” in the *Transact-SQL Users Guide*.

- If the table is a heap, round-robin-partitioned table, the random nature of heap table inserts should keep partitions balanced.

Take care with large bulk copy in operations. However, if the table is a semantic partitioned table, consider changing the partition condition using `alter table... partition by` for appropriate load balance. You can use parallel bulk copy (`parallel bcp`) to send rows to the partition with the smallest number of pages to balance the data across the partitions. See Chapter 4, “Using bcp to Transfer Data to and from SAP ASE,” in the *Utility Guide*.

1.5.3 Tables with Random Data Modification

Tables with clustered indexes that experience many inserts, updates, and deletes over time tend to lead to data pages that are approximately 70 to 75% full. This can lead to performance degradation in several ways.

- More pages must be read to access a given number of rows, requiring additional I/O and wasting data cache space.
- On tables that use allpages locking, the performance of large I/O and asynchronous prefetch suffers because the page chain crosses extents and allocation units.

Buffers brought in by large I/O may be flushed from cache before all of the pages are read. The asynchronous prefetch look-ahead set size is reduced by cross-allocation unit hops while following the page chain.

For tables that use data-only locking, large I/O and asynchronous prefetch performance suffers because the forward pages cross extents and allocation units.

Once the fragmentation starts to degrade on application performance, perform maintenance, keeping in mind that dropping and recreating clustered indexes requires 120% of the space occupied by the table.

If space is unavailable, maintenance becomes more complex and takes longer. The best, and often cheapest, solution is to add enough disk capacity to provide room for the index creation.

1.6 Disks and Full Devices

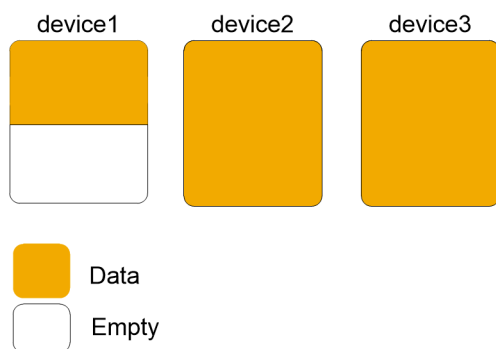
Simply adding disks and recreating indexes when partitions are full may not solve load-balancing problems. If a physical device that holds a partition becomes completely full, the data-copy stage of recreating an index cannot copy data to that physical device.

If a physical device is almost completely full, recreating the clustered index does not always succeed in establishing a good load balance.

1.6.1 Add Disks When Devices are Full

The result of creating a clustered index when a physical device is completely full is that two partitions are created on one of the other physical devices.

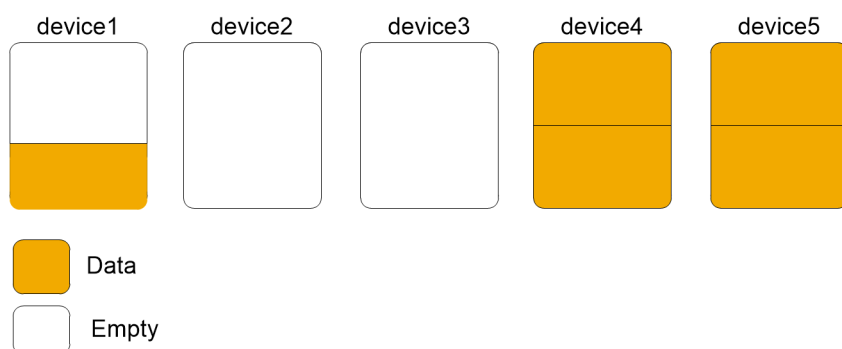
In this example, devices2 and device3 are completely full.



Adding two devices, repartitioning the table to use five partitions, and dropping and recreating the clustered index produces the following results:

Device 1	One partition, approximately 40% full.
Devices 2 and 3	Empty. These devices had no free space when <code>create index</code> started, so a partition for the copy of the index cannot be created on the device.
Devices 4 and 5	Each device has two partitions, and each is 100% full.

The results are:

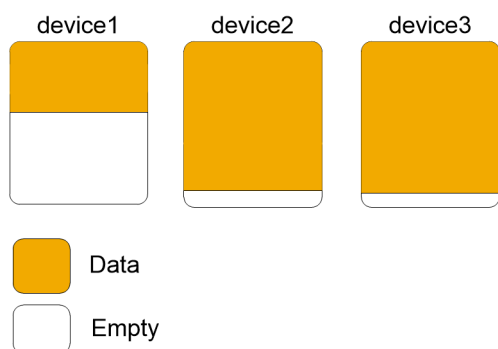


The only solution, once a device becomes completely full, is to bulk-copy the data out, truncate the table, and copy the data into the table again.

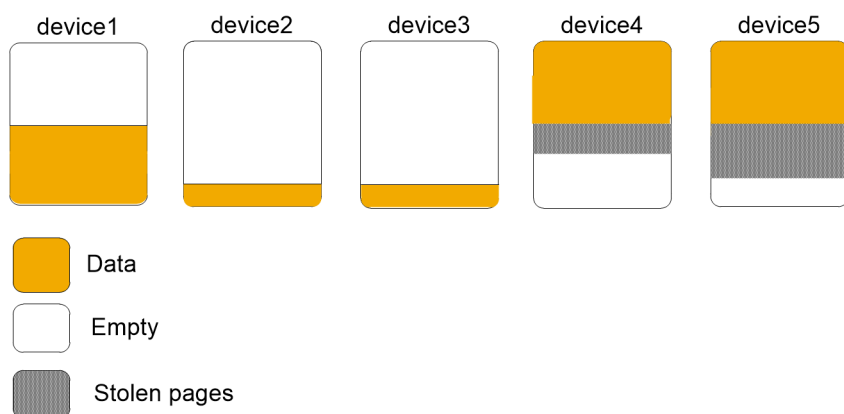
1.6.2 Add Disks When Devices are Nearly Full

If a device is nearly full, recreating a clustered index does not balance data across devices. Instead, the device that is nearly full stores a small portion of the partition, and the other space allocations for the partition steals extents on other devices.

This example shows a table with nearly full data devices.



After adding devices and recreating the clustered index, the result might be similar to the results:



Once the partitions on `device2` and `device3` use the small amount of space available, they start stealing extents from `device4` and `device5`.

Re-creating the indexes another time may lead to a more balanced distribution. However, if one of the devices is nearly filled by extent stealing, re-creating indexes again does not solve the problem.

Using bulk copy to copy the data out and back in again is most effective solution to this form of imbalance.

To avoid situations such as these, monitor space usage on the devices, and add space early.

1.7 Maintenance Issues and Partitioned Tables

Maintenance activity requirements for partitioned tables depend on the frequency and type of updates performed on the table.

Partitioned tables that require little maintenance include:

- Tables that are read-only or that experience very few updates. For tables that have few updates, only periodic checks for balance are required.
- Tables where inserts are well-distributed across the partitions. Random inserts to partitioned heap tables and inserts that are evenly distributed due to a clustered index key that places rows on different partitions do not develop skewed distribution of pages.
If data modifications lead to space fragmentation and partially filled data pages, you may need to recreate the clustered index.
- Heap tables where inserts are performed by bulk copy. You can use parallel bulk copy to direct the new data to specific partitions to maintain load balancing.

Partitioned tables that require frequent monitoring and maintenance include tables with clustered indexes that tend to direct new rows to a subset of the partitions. An ascending key index is likely to require more frequent maintenance.

1.7.1 Regular Maintenance Checks for Partitioned Tables

Routine monitoring for partitioned tables should include certain types of checks, in addition to routine database consistency checks.

- Use `sp_helppartition` to check the balance on partitions. If some partitions are significantly larger or smaller than the average, recreate the clustered index to redistribute data.
- Use `sp_helpsegment` to check the balance of space on underlying disks.
- If you recreate the clustered index to redistribute data for parallel query performance, check for devices that are nearing 50% full. Adding space before devices become too full avoids the complicated procedures described earlier in this chapter.
- Use `sp_helpsegment` to check the space available as free pages on each device, or use `sp_helppdb` to check for free kilobytes.

You might need to recreate the clustered index on partitioned tables because:

- Your index key tends to assign inserts to a subset of the partitions.
- Delete activity tends to remove data from a subset of the partitions, leading to I/O imbalance and partition-based scan imbalances.
- The table has many inserts, updates, and deletes, leading to many partially filled data pages. This condition leads to wasted space, both on disk and in the cache, and increases I/O because more pages need to be read for many queries.

2 Data Storage

Understand how to improve SAP ASE performance by creating indexes, tuning queries, and addressing object storage issues.

2.1 Query Optimization

The SAP ASE optimizer attempts to find the most efficient access path to your data for each table in a query by estimating the cost of the physical I/O needed to access the data, and the number of times each page must be read while in the data cache.

In most database applications, there are many tables in the database, and each table has one or more indexes. Depending on whether you have created indexes, and what kind of indexes you have created, the optimizer's access method options include:

- Table scan – reading all the table's data pages, sometimes hundreds or thousands of pages.
- Index access – using the index to find only the data pages needed, sometimes as few as three or four page reads in all.
- Index covering – using only an index to return data, without reading the actual data rows, requiring only a fraction of the page reads required for a table scan.

Using the appropriate indexes on tables should allow most queries to access the data they need with a minimum number of page reads.

2.1.1 Query Processing and Page Reads

Most of a query's execution time is spent reading data pages from disk. Therefore, most performance improvement comes from reducing the number of disk reads needed for each query.

When a query performs a table scan, SAP ASE reads every page in the table because no indexes are available to help it retrieve the data. The query may have poor response time, because disk reads take time. Queries that incur costly table scans also affect the performance of other queries on your server.

Table scans can increase the time other users have to wait for a response, since they use system resources such as CPU time, disk I/O, and network capacity.

Table scans use a large number of disk reads (I/Os) for a given query. When you have become familiar with the access methods, tuning tools, the size and structure of your tables, and the queries in your applications, you should be able to estimate the number of I/O operations a given join or select operation will perform, given the indexes that are available.

If you know what the indexed columns on your tables are, and the table and index sizes, you can often look at a query and predict its behavior. For different queries on the same table, you might be able to draw these conclusions:

- This query returns a single row or a small number of rows that match the `where` clause condition. The condition in the `where` clause is indexed; it should perform two to four I/Os on the index and one more to read the correct data page.
- All columns in the select list and `where` clause for this query are included in a nonclustered index. This query will probably perform a scan on the leaf level of the index, about 600 pages. Adding an unindexed column to the select list would force the query to scan the table, which would require 5000 disk reads.
- No useful indexes are available for this query; it is going to do a table scan, requiring at least 5000 disk reads.

This chapter describes how tables are stored, and how access to data rows takes place when indexes are not being used.

Chapter 5, “Indexes,” in *Performance and Tuning Series: Locking and Concurrency Control* describes access methods for indexes. This provides a basis for understanding how the optimizer models the cost of accessing the data for your queries.

Related Information

[Setting Space Management Properties \[page 50\]](#)

[Table and Index Size \[page 71\]](#)

2.2 SAP ASE Pages

Certain types of pages store database objects.

They are:

- Data pages – store the data rows for a table.
- Index pages – store the index rows for all levels of an index.
- Large object (LOB) pages – store the data for `text` and `image` columns, and for Java off-row columns.

SAP ASE versions 12.5 and later do not use the `buildmaster` binary to build the master device. Instead, SAP® has incorporated the `buildmaster` functionality in the `dataserver` binary.

The `dataserver` command allows you to create master devices and databases with logical pages of size 2K, 4K, 8K, or 16K. Larger logical pages allow you to create larger rows, which can improve your performance because SAP ASE accesses more data each time it reads a page. For example, a 16K page can hold 8 times the amount of data as a 2K page, an 8K page holds 4 times as much data as a 2K page, and so on, for all the sizes for logical pages.

The logical page size is a server-wide setting; you cannot have databases with varying size logical pages within the same server. All tables are automatically appropriately sized so that the row size is no greater than the current page size of the server. That is, rows cannot span multiple pages.

See the *Utility Guide* for specific information about using the `dataserver` command to build your master device.

SAP ASE may be required to process large volumes of data for a single query, DML operation, or command. For example, if you use a data-only-locked (DOL) table with a `char(2000)` column, SAP ASE must allocate memory to perform column copying while scanning the table. Increased memory requests during the life of a query or command means a potential reduction in throughput.

The size of SAP ASE logical pages determines the server's space allocation. Each allocation page, object allocation map (OAM) page, data page, index page, text page, and so on is built on a logical page. For example, if the logical page size of SAP ASE is 8K, each of these page types are 8K in size. All of these pages consume the entire size specified by the size of the logical page. OAM pages have a greater number of OAM entries for larger logical pages (for example, 8K) than for smaller pages (2K).

2.2.1 Page Headers and Page Sizes

All pages have a header that stores information, such as the partition ID that the page belongs to, and other information used to manage space on the page.

This table shows the number of bytes of overhead and usable space on data and index pages for a server configured for 2K pages.

Locking Scheme	Overhead	Bytes for User Data
Allpages	32	2016
Data-only	46	2002

The rest of the page is available to store data and index rows.

Related Information

[Large Object \(LOB\) Pages \[page 25\]](#)

2.2.2 Data and Index Pages

Data pages and index pages on data-only-locked tables have a row offset table that stores pointers to the starting byte for each row on the page. Each pointer takes 2 bytes.

Data and index rows are inserted on a page starting just after the page header, and fill in contiguously. For all tables and indexes on data-only-locked tables, the row offset table begins at the last byte on the page, and grows upward.

The information stored for each row consists of the actual column data, plus information such as the row number and the number of variable-length and null columns in the row. Index pages for allpages-locked tables do not have a row offset table.

Rows cannot cross page boundaries, except for `text`, `image`, and Java off-row columns. Each data row has at least 4 bytes of overhead; rows that contain variable-length data have additional overhead.

Related Information

[Table and Index Size \[page 71\]](#)

2.2.3 Large Object (LOB) Pages

`text`, `image`, and Java off-row columns for a table are stored as a separate data structure, consisting of a set of pages. These columns are known as large object, or LOB, columns.

Each table with a `text` or `image` column has one of these structures. If a table has multiple LOB columns, it still has only one of these separate data structures.

The table itself stores a 16-byte pointer to the first page of the value for the row. Additional pages for the value are linked by next and previous pointers. Each value is stored in its own separate page chain. The first page stores the number of bytes in the text value. The last page in the chain for a value is terminated with a null next-page pointer.

Reading or writing a LOB value requires at least two page reads or writes:

- One for the pointer
- One for the actual location of the text in the text object

The number of bytes each LOB page stores depends on the server's logical page size:

- 2k – 1800 bytes
- 4k – 3600 bytes
- 8k – 7650 bytes
- 16k – 16200 bytes

Every non-null value uses at least one full page.

LOB structures are listed separately in `sysindexes`.

The ID for the LOB structure is the same as the table's ID. The index ID column is `indid` and is always 255, and the `name` is the table name, prefixed with the letter "t".

2.2.4 Extents

SAP ASE pages are always allocated to a table, index, or LOB structure. A block of 8 pages is called an extent.

The size of an extent depends on the page size the server uses. The extent size on a 2K server is 16K; on an 8K it is 64K, and so on. The smallest amount of space that a table or index can occupy is 1 extent, or 8 pages. Extents are deallocated only when all the pages in an extent are empty.

The use of extents in SAP ASE is transparent to the user except when examining reports on space usage. For example, reports from `sp_spaceused` display the space allocated (the `reserved` column) and the space used by data and indexes. The `unused` column displays the amount of space in extents that are allocated to an object, but not yet used to store data.

```
sp_spaceused titles
```

name	rowtotal	reserved	data	index_size	unused
titles	5000	1392 KB	1250 KB	94 KB	48 KB

In this report, the `titles` table and its indexes have 1392KB reserved on various extents, including 48KB (24 data pages) that is unallocated.

i Note

SAP ASE avoids wasting extra space by filling up existing allocated extents in the target allocation page, even though these extents are assigned to other partitions. The net effect is that extents are allocated only when there are no free extents in the target allocation page.

2.3 Pages that Manage Space Allocation

In addition to data, index, and LOB pages used for data storage, SAP ASE uses other types of pages to manage storage, track space allocation, and locate database objects. The `sysindexes` table also stores pointers that are used during data access.

The pages that manage space allocation and the `sysindexes` pointers are used to:

- Speed the process of finding objects in the database.
- Speed the process of allocating and deallocating space for objects.
- Provide a means for SAP ASE to allocate additional space for an object that is near the space already used by the object. This helps performance by reducing disk-head travel.

These pages track the disk space use by database objects:

- Global allocation map (GAM) pages contain allocation bitmaps for an entire database.
- Allocation pages track space usage and objects within groups of 256 pages, or .5MB.
- Object allocation map (OAM) pages contain information about the extents used for an object. Each partition of a table and index has at least one OAM page that tracks where pages for the object are stored in the database.
- OAM pages manage space allocation for partitioned tables.

2.3.1 Global Allocation Map (GAM) Pages

Each database has a GAM, which stores a bitmap for all allocation units of a database, with 1 bit per allocation unit. When an allocation unit has no free extents available to store objects, the corresponding bit in the GAM is set to 1.

This mechanism expedites allocating new space for objects. Users cannot view the GAM page; it appears in the system catalogs as the `sysgams` table.

2.3.2 Allocation Pages

When you create a database or add space to a database, the space is divided into allocation units of 256 data pages. The first page in each allocation unit is the allocation page. Page 0 and all pages that are multiples of 256 are allocation pages.

The allocation page tracks space in each extent on the allocation unit by recording the partition ID, object ID, and index ID for the object that is stored on the extent, and the number of used and free pages. The allocation page also stores the page ID for the table or index's corresponding OAM page.

2.3.3 Object Allocation Map Pages

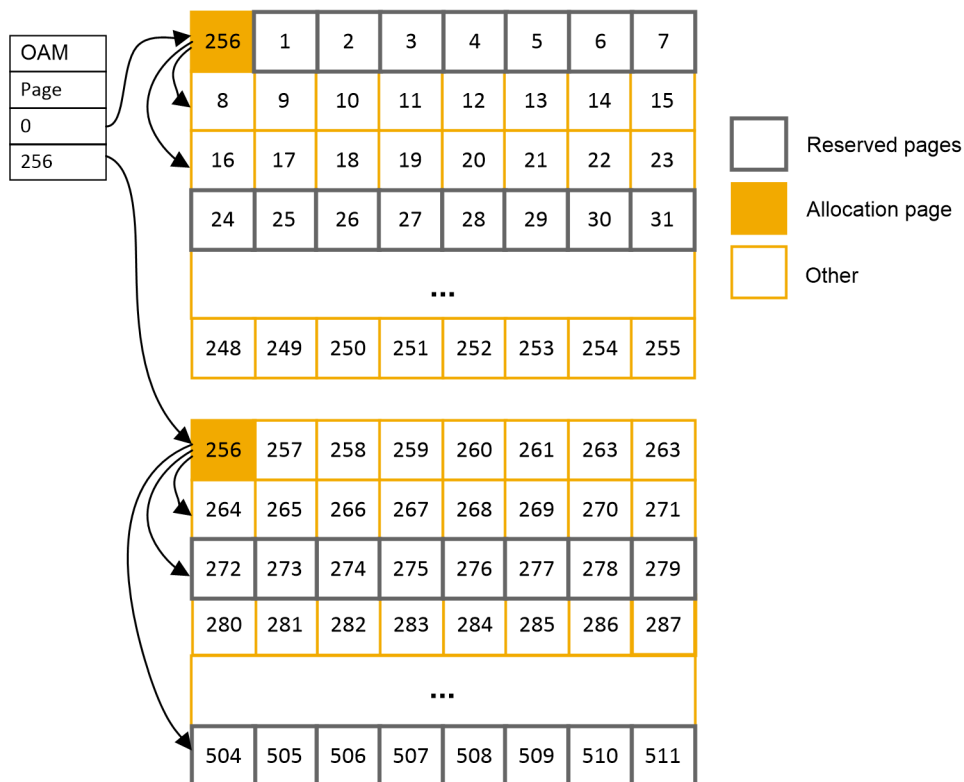
Each partition for a table, index, and text chain has one or more object allocation map (OAM) pages stored on pages allocated to the table or index.

If a partition has more than one OAM page, the pages are linked in a chain. OAM pages store pointers to the allocation units that contain pages for the object.

The first page in the chain stores allocation hints, indicating which OAM page in the chain stores information about allocation units with free space. This provides a fast way to allocate additional space for an object and to keep the new space close to pages already used by the object.

2.3.4 How OAM Pages and Allocation Pages Manage Object Storage

An illustration showing how allocation units, extents, and objects are managed by OAM pages and allocation pages.



- Two allocation units are shown, one starting at page 0 and one at page 256. The first page of each is the allocation page.
- A table is stored on four extents, starting at pages 1 and 24 on the first allocation unit and pages 272 and 504 on the second unit.
- The first page of the table is the table's OAM page. It points to the allocation page for each allocation unit where the object uses pages, so it points to pages 0 and 256.
- Allocation pages 0 and 256 store the table's object ID, index ID, and partition ID to which the extent belongs. Allocation page 0 points to pages 1 and 24 for the table, and allocation page 256 points to pages 272 and 504.

2.3.5 Page Allocation Keeps an Object's Pages Together

SAP ASE attempts to keep page allocations close together for each object (for example, a table partition, index, or a table's text or image chain).

Typically, when SAP ASE requires a new page:

- If the object's current extent contains a free page, SAP ASE uses this page.
- If the current extent has no free pages, but another extent assigned to the object on the same allocation unit has a free page, SAP ASE uses this page.
- If the current allocation unit has no extents with free pages assigned to the object, but has a free extent, Adaptive Server allocates the first available page of the free extent.
- If the current allocation unit is full, SAP ASE scans the object's OAM page for another allocation unit containing extents with free pages, and uses the first available page.

- If no OAM entries indicate available free pages, SAP ASE compares the OAM entries to the global allocation map page to see if any allocation units have free extents. SAP ASE allocates the first available page of first free extent.
- If all of the OAM entries are for full allocation units, SAP ASE searches the global allocation map for an allocation unit with at least one free extent. SAP ASE adds a new OAM entry for that allocation unit to the object's OAM, allocates a free extent, and uses the first free page on the extent.

i Note

Operations like `bcp` and `reorg rebuild` using large scale allocation do not look for free pages on already allocated extents; instead they allocate full free extents. Large scale allocation cannot typically use the first extent of each allocation unit. The first extent only has 7 usable pages because its first page holds the allocation page structure.

2.3.6 Data Access Using `sysindexes` and `syspartitions`

The `sysindexes` table stores information about indexed and nonindexed tables. `syspartitions` stores information about each table and index partition and includes one row per partition.

`sysindexes` has one row for each:

- Allpages-locked table – the `indid` column is 0 if the table does not have a clustered index, and 1 if the table does have a clustered index.
- Data-only-locked tables – the `indid` is always 0 for the table.
- Nonclustered index – and for each clustered index on a data-only-locked table.
- Table with one or more LOB columns – the index ID is always 255 for the LOB structure.

In SAP ASE version 15.0. and later, each row in `syspartitions` stores pointers to a table or index to speed access to objects.

Table 1: Use of `syspartitions` Pointers in Data Access

Column	Use for Table Access	Use for Index Access
<code>rootpage</code>	If <code>indid</code> is 0 and the table is a partitioned allpages-locked table, <code>root</code> points to the last page of the heap.	Used to find the root page of the index tree.
<code>firstpage</code>	Points to the first data page in the page chain for allpages-locked tables.	Points to the first leaf-level page in a nonclustered index, or a clustered index on a data-only-locked table.
<code>datoampage</code>	Points to the first OAM page for the table.	
<code>indoampage</code>		Points to the first OAM page for an index.

2.4 Space Overheads

Regardless of the logical page size for which it is configured, SAP ASE allocates space for objects (tables, indexes, text page chains) in extents, each of which is eight logical pages.

That is, if a server is configured for 2K logical pages, it allocates one extent, 16K, for each of these objects; if a server is configured for 16K logical pages, it allocates one extent, 128K, for each of these objects.

This is also true for system tables. If your server has many small tables, space consumption can be quite large if the server uses larger logical pages.

For example, for a server configured for 2KB logical pages, `systypes`—with approximately 31 short rows, a clustered and a non-clustered index—reserves 3 extents, or 48KB of memory. If you migrate the server to use 8KB pages, the space reserved for `systypes` is still 3 extents, 192KB of memory.

For a server configured for 16KB, `systypes` requires 384KB of disk space. For small tables, the space unused in the last extent can become significant on servers using larger logical page sizes.

Databases are also affected by larger page sizes. Each database includes the system catalogs and their indexes. If you migrate from a smaller to larger logical page size, you must account for the amount of disk space each database requires.

2.4.1 Number of Columns and Size

Column and row size limits.

The maximum number of columns you can create in a table is:

- 1024 for fixed-length columns in both allpages-locked (APL) and data-only-locked (DOL) tables
- 254 for variable-length columns in an APL table
- 1024 for variable-length columns in an DOL table

The maximum size of a column depends on:

- Whether the table includes variable- or fixed-length columns.
- The logical page size of the database. For example, in a database with 2K logical pages, the maximum size of a column in an APL table can be as large as a single row, about 1962 bytes, less the row format overheads. Similarly, for a 4K page, the maximum size of a column in a APL table can be as large as 4010 bytes, less the row format overheads.
- If you attempt to create a table with a fixed-length column that is greater than the limits of the logical page size, `create table` issues an error message.

Locking Scheme	Page Size	Maximum Row Length	Maximum Column Length
	2K (2048 bytes)	1962	1960 bytes
	4K (4096 bytes)	4010	4008 bytes
APL tables	8K (8192 bytes)	8106	8104 bytes

Locking Scheme	Page Size	Maximum Row Length	Maximum Column Length
	16K (16384 bytes)	16298	16296 bytes
	2K (2048 bytes)	1964	1958 bytes
	4K (4096 bytes)	4012	4006 bytes
DOL tables	8K (8192 bytes)	8108	8102 bytes
	16K (16384 bytes)	16300	16294 bytes if table does not include any variable-length columns
	16K (16384 bytes)	16300 Subject to a max start offset of varlen = 8191, unless the database is configured with <code>allow wide dol rows</code> enabled.	8191-6-2 = 8183 bytes if table includes at least one variable-length column.

The maximum size of a fixed-length column in a DOL table with a 16K logical page size depends on whether the table contains variable-length columns. The maximum possible starting offset of a variable-length column is 8191. If the table has any variable-length columns, the sum of the fixed-length portion of the row, plus overheads, cannot exceed 8191 bytes, and the maximum possible size of all the fixed-length columns is restricted to 8183 bytes, when the table contains any variable-length columns.

In the table, the maximum column length is determined by subtracting 6 bytes for row overhead and 2 bytes for row length field.

2.4.1.1 Variable-Length Columns in APL Tables

APL tables that contain one variable-length column (for example, `varchar`, `varbinary` and so on) have a specific minimum size of overhead for each row.

- 2 bytes for the initial row overhead.
- 2 bytes for the row length.
- 2 bytes for the column-offset table at the end of the row. This is always `<n+1>` bytes, where `<n>` is the number of variable-length columns in the row.

A single-column table has an overhead of at least 6 bytes, plus additional overhead. The maximum column size, after overhead, must be less than or equal to: (column length) + (additional overhead) + (six-byte overhead.)

Table 2: Maximum Size for Variable-length Columns in an APL Table

Page Size	Maximum Row Length	Maximum Column Length
2K (2048 bytes)	1962	1948

Page Size	Maximum Row Length	Maximum Column Length
4K (4096 bytes)	4010	3988
8K (8192 bytes)	8096	8068
16K (16384 bytes)	16298	16228

2.4.1.1.1 Variable-Length Columns that Exceed the Logical Page Size

If your table uses 2K logical pages, you can create some variable-length columns whose total row-length exceeds the maximum row-length for a 2K page size. This allows you to create tables where some, but not all, variable-length columns contain the maximum possible size.

However, when you issue `create table`, you receive a warning message that says the resulting row size may exceed the maximum possible row size, and cause a future `insert` or `update` to fail.

For example, if you create a table that uses a 2K page size, and contains a variable-length column with a length of 1975 bytes, SAP ASE creates the table but issues a warning message. You cannot insert data that exceeds the maximum length of the row (1962 bytes).

2.4.1.2 Variable-Length Columns in DOL Tables

Maximum size for variable-length columns in an DOL table.

For a single, variable-length column in a DOL table, the minimum overhead for each row is:

- Six bytes for the initial row overhead.
- Two bytes for the row length.
- Two bytes for the column offset table at the end of the row. Each column offset entry is two bytes. There are `<n>` such entries, where `n` is the number of variable-length columns in the row.

The total overhead is 10 bytes. There is no adjust table for DOL rows. The actual variable-length column size is:

```
column length + 10 bytes overhead
```

Page Size	Maximum Row Length	Maximum Column Length
2K (2048 bytes)	1964	1954
4K (4096 bytes)	4012	4002
8K (8192 bytes)	8108	8098
16K (16384 bytes)	16300	16290

In databases using a 16k page size that do not have `allow wide dol rows` enabled, DOL tables with variable-length columns must have an offset of fewer than 8191 bytes for all inserts to succeed. For example, this insert fails because the offset for columns `c2`, `c3`, and `c4` is 9010, which exceeds the maximum of 8191 bytes:

```
create table t1(
  c1 int not null,
  c2 varchar(5000) not null
  c3 varchar(4000) not null
  c4 varchar(10) not null
  ... more fixed length columns)
cvarlen varchar(nnn) lock datarows
```

2.4.1.2.1 Wide, Variable-Length Rows

SAP ASE allows data-only locked (DOL) columns to use a row offset of up to 16332 bytes for wide, variable-length DOL rows if it is configured for a logical page size of 16K.

Enable wide, variable-length DOL rows for each database using:

```
sp_dboption <database_name>, 'allow wide dol rows', true
```

i Note

`allow wide dol rows` is on by default for temporary databases. You cannot set `allow wide dol rows` for the master database.

`sp_dboption 'allow wide dol rows'` has no effect on user databases with logical page sizes smaller than 16K; SAP ASE cannot create a wide, variable-length DOL rows on pages smaller than 16384 bytes.

This example configures the `pubs2` database on a server using a page size of 16K to use wide, variable-length DOL rows:

1. Enable wide, variable-length rows for the `pubs2` database:

```
sp_dboption pubs2, 'allow wide dol rows', true
```

2. Create the `book_desc` table, which includes wide, variable-length DOL columns that begin after the row offset of 8192:

```
create table book_desc
(title varchar(80) not null,
title_id varchar(6) not null,
author_desc char(8192) not null,
book_desc varchar(5000) not null)
lock datarows
```

Bulk-copying Wide Data

You must use the version of `bcp` shipped with SAP ASE version 15.7 and later to bulk-copy-in data that contains wide, variable-length DOL rows. You must configure the database receiving the data to accept wide, variable-

length DOL rows (that is, `bcp` does not copy wide rows into databases for which you have not enabled `allow wide dol rows`).

Dumping and Loading Wide, Variable-length DOL Columns

Database and transaction log dumps retain their setting for `allow wide dol rows`, which is imported into the database into which you are loading the dump (if this database does not already have the option set).

For example, if you load a transaction log dump named `my_table_log.dmp`, which has `allow wide dol rows` set to `true`, into database `big_database`, for which you have not set `allow wide dol rows`, `my_table.log` retains its setting of `true` for `allow wide dol rows` after the load to `big_database` completes.

However, if the database or transaction log dump does not have `allow wide dol rows` set, but the database into which you are loading the dump does, `allow wide dol rows` remains set.

Using Proxy Tables with Wide, Variable-length DOL Rows

You can use proxy tables with wide, variable-length DOL rows.

When you create proxy tables (regardless of their row length), the controlling SAP ASE is the one on which you execute the `create table` or `create proxy table` command. SAP ASE executes these commands on the server to which you are connected. However, SAP ASE executes data manipulation statements (`insert`, `delete`) on the server on which the data is stored, and the user's local server only formats the request and then sends it; the local server does not control anything.

SAP ASE creates proxy tables as though they are created on the local server, even though the data resides on remote servers. If the `create proxy table` command creates a DOL table that contains wide, variable-length rows, the command succeeds only if the database in which you are creating the proxy table has `allow wide dol rows` set to `true`.

i Note

SAP ASE creates proxy tables using the local server's `lock scheme` configuration, and creates a proxy table with DOL rows if `lock scheme` is set to `datarows` or `datapages`.

When you insert or update data in proxy tables, SAP ASE ignores the local database's setting for `allow wide dol rows`. The server where the data resides determines whether an `insert` or `update` succeeds

2.4.1.2.2 Restrictions for Converting Locking Schemes or Using `select into`

Certain restrictions apply whether you are using `alter table` to change a locking scheme or using `select into` to copy data into a new table.

For servers that use page sizes other than 16K pages, the maximum length of a variable-length column in an APL table is less than that for a DOL table, so you can convert the locking scheme of an APL table with a maximum sized variable-length column to DOL. You cannot, however, convert a DOL table containing at least one maximum sized variable-length column to an APL table.

On servers that use 16K pages, APL tables can store substantially larger sized variable-length columns than DOL tables. You can convert tables from DOL to APL, but you cannot convert from APL to DOL.

These restrictions on locking-scheme conversions occur only if data in the source table exceeds the limits of the target table. If this occurs, SAP ASE raises an error message while transforming the row format from one locking scheme to the other. If the table is empty, no such data transformation is required, and the lock-change operation succeeds. However, subsequent `inserts` or `updates` to the table, users may see errors caused by limitations on the column or row size for the target schema of the altered table.

2.4.1.2.3 Organizing Columns in DOL Tables by Size of Variable-length Columns

For DOL tables that use variable-length columns, arrange the columns so that the longest columns are placed toward the end of the table definition. This allows you to create tables with much larger rows than if the large columns appear at the beginning of the table definition.

For instance, in a 16K page server, the following table definition is acceptable:

```
create table t1 (  
  c1 int not null,  
  c2 varchar(1000) null,  
  c3 varchar(4000) null,  
  c4 varchar(9000) null) lock datarows
```

However, the following table definition typically is unacceptable for future inserts. The potential start offset for column `c2` is greater than the 8192-byte limit because of the preceding 9000-byte `c4` column:

```
create table t2 (  
  c1 int not null,  
  c4 varchar(9000) null,  
  c3 varchar(4000) null,  
  c2 varchar(1000) null) lock datarows
```

The table is created, but future inserts may fail.

2.4.2 Number of Rows per Data Page

Maximum number of data rows for a DOL data page.

The number of rows allowed for a DOL data page is determined by:

- The page size
- A 10-byte overhead for the row ID, which specifies a row-forwarding address

Page size	Maximum number of rows
2K	166
4K	337
8K	678
16K	1361

APL data pages can have a maximum of 256 rows. Because each page requires a one-byte row number specifier, large pages with short rows incur some unused space. For example, if SAP ASE is configured with 8K logical pages and rows that are 25 bytes, each page has 1275 bytes of unused space, after accounting for the row-offset table and the page header.

2.4.3 Additional Number of Object and Size Restrictions

Maximum number of arguments for stored procedures and SAP ASE version considerations for data limits.

The maximum number of arguments for stored procedures is 2048. See Chapter 16, “Using Stored Procedures,” in the *Transact-SQL Users Guide*.

SAP ASE version 12.5 and later can store data that has different limits than data stored in versions earlier than 12.5. Clients must be able to store and process these newer data limits. If you are using older versions of Open Client and Open Server, they cannot process the data if you:

- Upgrade to SAP ASE version 12.5 or later
- Drop and recreate the tables with wide columns
- Insert wide data

See Chapter 2, “Basic Configuration for Open Client” in the *Open Client Configuration Guide*.

2.5 Tables without Clustered Indexes

If you create a table on SAP ASE, but do not create a clustered index, the table is stored as a heap, which means the data rows are not stored in any particular order.

`select`, `insert`, `delete`, and `update` operations perform on heap tables when there is no “useful” index to aid in retrieving data.

There are very few justifications for heap tables. Most applications perform better with clustered indexes on the tables. However, heap tables work well for small tables that use only a few pages, and for tables where changes are infrequent

Heap tables can be useful for tables that do not require:

- Direct access to single, random rows
- Ordering of result sets

Heap tables do not work well for queries against most large tables that must return a subset of the table's rows.

Partitioned heap tables are useful in applications with frequent, large volumes of batch inserts where the overhead of dropping and creating clustered indexes is unacceptable.

Sequential disk access is efficient, especially with large I/O and asynchronous prefetch. However, the entire table must always be scanned to find any value, and this has potentially large impact in the data cache and other queries.

Batch inserts can also perform efficient sequential I/O. However, there is a potential bottleneck on the last page if multiple processes try to insert data concurrently.

Sometimes, an index exists on the columns named in a `where` clause, but the optimizer determines that it would be more costly to use the index than to perform a table scan.

Table scans are always used when you select all rows in a table. The only exception is when the query includes only columns that are keys in a nonclustered index.

For more information, see Chapter 5, "Indexes," in *Performance and Tuning Series: Locking and Concurrency Control*.

2.5.1 Lock Schemes

The data pages in an APL table are linked into a list of pages by pointers on each page. Pages in data-only-locked tables are not linked into a page chain.

In an allpages-locked table, each page stores a pointer to the next page in the chain and to the previous page in the chain. When you insert new pages, the pointers on the two adjacent pages change to point to the new page. When SAP ASE scans an allpages-locked table, it reads the pages in order, following these page pointers.

Pages are also doubly linked at each index level of allpages-locked tables, and at the leaf level of indexes on data-only-locked tables. If an allpages-locked table is partitioned, there is one page chain for each partition.

Unlike allpages-locked tables, data-only-locked tables typically do not maintain a page chain, except immediately after you create a clustered index. However, this page chain is broken the first time you issue a command on the table.

When SAP ASE scans a data-only-locked table, it uses the information stored in the OAM pages.

Another difference between allpages-locked tables and data-only-locked tables is that data-only-locked tables use fixed row IDs. This means that row IDs (a combination of the page number and the row number on the page) do not change in a data-only-locked table during normal query processing.

Row IDs change only when one of the operations that require data-row copying is performed, for example, during `reorg rebuild` or while creating a clustered index.

Related Information

[Object Allocation Map Pages \[page 27\]](#)

[Deletion from a Data-Only locked Heap Table \[page 40\]](#)

[Data-Only-Locked Heap Tables \[page 41\]](#)

2.5.2 select Operations on Heap Tables

When you issue a `select` query on a heap, and there is no useful index, SAP ASE must scan every data page in the table to find every row that satisfies the conditions in the query. There may be one row, many rows, or no rows that match.

2.5.2.1 Allpages-locked Heap Tables

For allpages-locked tables, SAP ASE reads the `firstpage` column in `syspartitions` for the table, reads the first page into cache, and follows the next page pointers until it finds the last page of the table.

2.5.2.2 Data-Only Locked Heap Tables

Since the pages of data-only-locked tables are not linked in a page chain, a `select` query on a data-only-locked heap table uses the table's OAM and the allocation pages to locate all the rows in the table.

The OAM page points to the allocation pages, which point to the extents and pages for the table.

2.5.3 Data Insertion into an Allpages-Locked Heap Table

When you insert data into an allpages-locked heap table without a clustered index, the data row is always added to the last page of the table.

If there is no clustered index on a table, and the table is not partitioned, the `syspartitions.root` entry for the heap table stores a pointer to the last page of the heap to indicate the page where the data should be inserted.

If the last page is full, a new page is allocated in the current extent and linked onto the chain. If the extent is full, SAP ASE looks for empty pages on other extents being used by the table. If no pages are available, a new extent is allocated to the table.

One of the severe performance limits on heap tables that use allpages locking is that the page must be locked when the row is added, and the lock is held until the transaction completes. If many users are trying to insert into an allpages-locked heap table simultaneously, each insert must wait for the preceding transaction to complete.

This problem of last-page conflicts on heap tables is true for:

- Single-row inserts
- Multiple row inserts using `select into` or `insert...select`, or several `insert` statements in a batch
- Bulk copying into the table

To address last-page conflicts on heap tables, try:

- Switching to datapages or datarows locking
- Creating a clustered index that directs the inserts to different pages
- Partitioning the table, which creates multiple insert points for the table, giving you multiple “last pages” in an allpages-locked table

For all transactions where there may be lock conflicts, you can also:

- Keep transactions short
- Avoid network activity and user interaction whenever possible, once a transaction acquires locks

2.5.4 Data Insertion into a Data-Only-Locked Heap Table

When users insert data into a data-only-locked heap table, SAP ASE tracks page numbers where the inserts have recently occurred, and keeps the page number as a suggestion for future tasks that need space. Subsequent inserts to the table are directed to one of these pages.

If the page is full, SAP ASE allocates a new page and replaces the old hint with the new page number.

Blocking while many users are simultaneously inserting data is much less likely to occur during inserts to data-only-locked heap tables than in APL tables. When blocking does occur, SAP ASE allocates a small number of empty pages and directs new inserts to those pages using these newly allocated pages as hints.

For datarows-locked tables, blocking occurs only while the actual changes to the data page are being written; although row locks are held for the duration of the transaction, other rows can be inserted on the page. The row-level locks allow multiple transaction to hold locks on the page.

There may be slight blocking on data-only-locked tables, because SAP ASE allows a small amount of blocking after many pages have just been allocated, so that the newly allocated pages are filled before additional pages are allocated.

Conflicts during inserts to heap tables are greatly reduced for data-only-locked tables, but can still take place. If these conflicts slow inserts, try:

- Switching to datarows locking, if the table uses datapages locking
- Using a clustered index to spread data inserts
- Partitioning the table, which provides additional hints and allows new pages to be allocated on each partition when blocking takes place

2.5.5 Data Deletion from a Heap Table

When you delete rows from a heap table that does not have a useful index, SAP ASE scans the data rows in the table to find the rows to delete. It cannot determine how many rows match the conditions in the query without examining every row.

2.5.5.1 Deletion from an Allpages-Locked Heap Table

When a data row is deleted from a page in an allpages-locked table, the rows that follow it on the page move up so that the data on the page remains contiguous, avoiding fragmentation within the page.

2.5.5.2 Deletion from a Data-Only locked Heap Table

When you delete rows from a data-only-locked heap table, a table scan is required if there is no useful index. The OAM and allocation pages are used to locate the pages.

The space on the page is not recovered immediately. Rows in data-only-locked tables must maintain fixed row IDs, and must be reinserted in the same place if the transaction is rolled back.

After a delete transaction commits, one of the following processes shifts rows on the page to make the space usage contiguous:

- The housekeeper garbage collection process
- An insert that needs to find space on the page
- The `reorg reclaim_space` command

2.5.5.3 Last Row Deletion on a Page

If you delete the last row on a page, the page is deallocated. If other pages on the extent are still in use by the table, the page can be used again by the table when a page is needed.

If all other pages on the extent are empty, the entire extent is deallocated. It can be allocated to other objects in the database. The first data page for a table or an index is never deallocated.

2.5.6 Update Data on a Heap Table

Like other operations on heap tables, an `update` on a table that has no useful index on the columns in the `where` clause performs a table scan to locate the rows to be changed.

2.5.6.1 Allpages-Locked Heap Tables

You can perform updates on allpages-locked heap tables in several ways.

- If the length of the row does not change, the updated row replaces the existing row, and no data moves on the page.
- If the length of the row changes, and there is enough free space on the page, the row remains in the same place on the page, but other rows move up or down to keep the rows contiguous on the page. The row offset pointers at the end of the page are adjusted to point to the changed row locations.
- If the row does not fit on the page, the row is deleted from its current page, and inserted as a new row on the last page of the table. This type of update may cause a conflict on the last page of the heap.

2.5.6.2 Data-Only-Locked Heap Tables

One of the requirements for data-only-locked tables is that the row ID of a data row never changes (except during intentional rebuilds of the table). Therefore, updates to data-only-locked tables can be performed by the first two methods described above, as long as the row fits on the page.

However, when a row in a data-only-locked table is updated so that it no longer fits on the page, a process called row forwarding performs these steps:

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

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

If a row must 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.

Row forwarding increases concurrency during update operations because indexes do not have to be updated. It can slow data retrieval, however, because a task must read the page at the original location and then read the page where the forwarded data is stored.

Use the `reorg` command to clear forwarded rows from a table.

See Chapter 1, “Understanding Query Processing” in *Performance and Tuning Series: Query Processing and Abstract Plans*.

2.5.7 How SAP ASE Performs I/O for Heap Operations

When a query needs a data page, SAP ASE first checks to see if the page is available in a data cache. If the page is not available, then it must be read from disk.

A newly installed SAP ASE with a 2K logical page size has a single data cache configured for 2K I/O. Each I/O operation reads or writes a single SAP ASE data page. A system administrator can:

- Configure multiple caches
- Bind tables, indexes, or text chains to the caches
- Configure data caches to perform I/O in page-sized multiples, up to eight data pages (one extent)

To use these caches most efficiently, and to reduce I/O operations, the SAP ASE optimizer can:

- Choose to prefetch up to eight data pages at a time
- Choose between different caching strategies

2.5.7.1 Sequential Prefetch, or Large I/O

SAP ASE data caches can be configured to allow large I/Os. When a cache allows large I/Os, SAP ASE can prefetch data pages.

Caches have buffer pools that depend on the logical page sizes, allowing SAP ASE to read up to an entire extent (eight data pages) in a single I/O operation.

Since much of the time required to perform I/O operations is taken up in seeking and positioning, reading eight pages in a 16K I/O takes nearly the same amount of time as a single page, 2K I/O. Reading eight pages using eight 2K I/Os is nearly eight times more costly than reading eight pages using a single 16K I/O. Table scans perform much better when you use large I/Os.

When several pages are read into cache with a single I/O, they are treated as a unit: they age in cache together, and if any page in the unit has been changed while the buffer was in cache, all pages are written to disk as a unit.

See Chapter 5, “Memory Use and Performance,” in *Performance and Tuning Series: Basics*.

i Note

Reference to large I/Os are on a 2K logical page size server. If you have an 8K page size server, the basic unit for the I/O is 8K. If you have a 16K page size server, the basic unit for the I/O is 16K.

2.5.8 Heap Table Maintenance

Over time, I/O on heap tables can become inefficient as storage becomes fragmented but you can reclaim space in heap tables.

Deletes and updates can result in:

- Many partially filled pages

- Inefficient large I/O, since extents may contain many empty pages
- Forwarded rows in data-only-locked tables

To reclaim space in heap tables:

- Use the `reorg rebuild` command (data-only-locked tables only)
- Create and then drop a clustered index
- Use `bcp` (the bulk copy utility) and `truncate table`

2.5.8.1 Use reorg rebuild to Reclaim Space

`reorg rebuild` copies all data rows to new pages and rebuilds any indexes on the heap table. You can use `reorg rebuild` on data-only-locked and allpages-locked tables. But you cannot use `reorg rebuild <table_name> <index_name>` or `<partition_name>` on allpages-locked tables.

`reorg rebuild` does not reorganize allocations for text and image data.

2.5.8.2 Reclaim Space by Creating a Clustered Index

To create a clustered index, you must have free space in the database that is at least 120% of the table size.

Related Information

[Determine the Space Available for Maintenance Activities \[page 110\]](#)

2.5.8.3 Reclaiming Space Using bcp

Reclaim space with `bcp`.

Procedure

1. Use `bcp` to copy the table to a file.
2. Use `truncate table` to truncate the table, reclaiming unused space.
3. Copy the table back in again with `bcp`.

Results

For detailed information about working with partitioned tables, see Chapter 10, “Partitioning Tables and Indexes,” in the *Transact-SQL Users Guide*.

For more information on `bcp`, see the *Utility Guide*.

2.5.9 Transaction Log: A Special Heap Table

The SAP ASE transaction log is a special heap table that stores information about data modifications in the database. The transaction log is always a heap table; each new transaction record is appended to the end of the log. The transaction log has no indexes.

Place logs on separate physical devices from the data and index pages. Since the log is sequential, the disk head on the log device rarely needs to perform seeks, and you can maintain a high I/O rate to the log.

Transaction log writes occur frequently. Do not let them contend with other I/O in the database, which usually happens at scattered locations on the data pages.

Besides recovery, these operations read the transaction log:

- Any data modification performed in deferred mode.
- Triggers that contain references to the inserted and deleted tables. These tables are built from transaction log records when the tables are queried.
- Transaction rollbacks.

In most cases, the transaction log pages for these queries are still available in the data cache when SAP ASE needs to read them, and disk I/O is not required.

Related Information

[Keep Transaction Logs on a Separate Disk \[page 11\]](#)

2.5.10 Asynchronous Prefetch and I/O on Heap Tables

Asynchronous prefetch speeds the performance of queries that perform table scans.

Any task that must perform a physical I/O relinquishes the server's engine (CPU) while it waits for the I/O to complete. If a table scan must read 1000 pages, and none of those pages are in cache, performing 2K I/O with no asynchronous prefetch means the task makes 1000 loops, executing on the engine, and then sleeping to wait for I/O. Using 16K I/O requires only 125 loops.

Asynchronous prefetch can request all of the pages on an allocation unit that belong to a table when the task fetches the first page from the allocation unit. If the 1000-page table resides on just 4 allocation units, the task requires many fewer cycles through the execution and sleep loops.

Type of I/O	Loops	Steps in Each Loop
2K I/O No prefetch	1000	<ol style="list-style-type: none"> 1. Request a page. 2. Sleep until the page has been read from disk. 3. Request a page. 4. Wait for a turn to run on the SAP ASE engine (CPU). 5. Read the rows on the page.
16K I/O No prefetch	125	<ol style="list-style-type: none"> 1. Request an extent. 2. Sleep until the extent has been read from disk. 3. Wait for a turn to run on the SAP ASE engine (CPU). 4. Read the rows on the eight pages.
Prefetch	4	<ol style="list-style-type: none"> 1. Request all the pages in an allocation unit. 2. Sleep until the first page has been read from disk. 3. Wait for a turn to run on the SAP ASE engine (CPU). 4. Read all the rows on all the pages in cache.

Actual performance depends on cache size and other activity in the data cache.

See Chapter 6, “Tuning Asynchronous Prefetch,” in *Performance and Tuning Series: Basics*.

2.6 Caches and Object Bindings

A table can be bound to a specific cache. If a table is not bound to a specific cache, but its database is bound to a cache, all of its I/O takes place in that cache.

Otherwise, the table’s I/O takes place in the default data cache. You can configure the default data cache for large I/O. Applications that use heap tables are likely to give the best performance when they use a cache configured for 16K I/O.

See Chapter 4, “Configuring Data Caches,” in the *System Administration Guide: Volume 2*.

2.6.1 Heap Tables, I/O, and Cache Strategies

Each SAP ASE data cache is managed as an MRU/LRU (most recently used/least recently used) chain of buffers. As buffers age in the cache, they move from the MRU end toward the LRU end.

When changed pages in the cache pass a point called the wash marker, on the MRU/LRU chain, SAP ASE initiates an asynchronous write on any pages that have changed while they were in cache. This helps ensure that when the pages reach the LRU end of the cache, they are clean and can be reused.

SAP ASE has two major strategies for using its data cache efficiently:

- LRU replacement strategy

- MRU, or fetch-and-discard replacement strategy

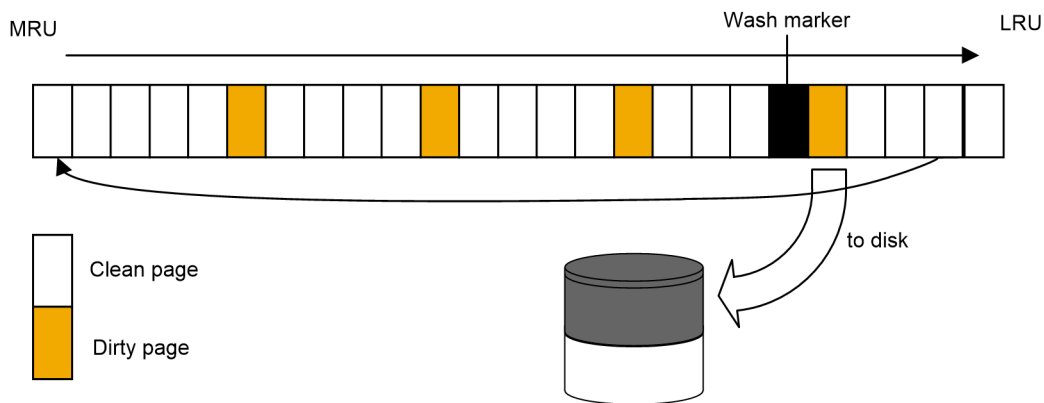
2.6.1.1 LRU Replacement Strategy

LRU replacement strategy reads the data pages sequentially into the cache, replacing a “least recently used” buffer. The buffer is placed on the MRU end of the data buffer chain. It moves toward the LRU end as more pages are read into the cache.

SAP ASE uses LRU strategy for:

- Statements that modify data on pages
- Pages that are needed more than once by a single query
- OAM pages
- Most index pages
- Any query where LRU strategy is specified

This figure illustrates how LRU strategy takes a clean page from the LRU end of the cache:



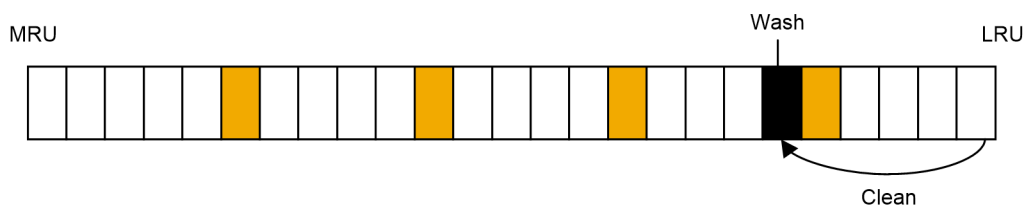
2.6.1.2 MRU Replacement Strategy

MRU replacement strategy is used for table scanning on heap tables.

MRU (fetch-and-discard) is most often used for queries where a page is needed only once by the query, including:

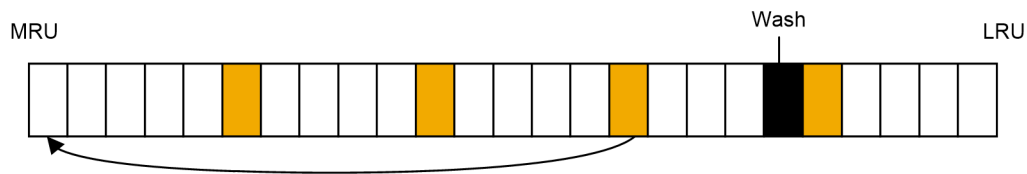
- Most table scans in queries that do not use joins
- One or more tables in a join query

This strategy places pages into the cache just before the wash marker:



Placing the pages needed only once at the wash marker means they do not push other pages out of the cache.

The fetch-and-discard strategy is used only on pages actually read from the disk for the query. If a page is already in cache due to earlier activity on the table, the page is placed at the MRU end of the cache.



2.6.2 select Operations and Caching

Under most conditions, single-table `select` operations on a heap use the largest I/O available to the table, and fetch-and-discard (MRU) replacement strategy.

For heap tables, select operations performing large I/O can be very effective. SAP ASE can read sequentially through all the extents in a table.

See Chapter 1, “Understanding Query Processing” in *Performance and Tuning Series: Query Processing and Abstract Plans*.

Unless the heap is being scanned as the inner table of a nested-loop join, data pages are needed only once for the query, so MRU replacement strategy reads and discards the pages from cache.

i Note

Large I/O on allpages-locked heap tables is effective only when the page chains are not fragmented.

Related Information

[Heap Table Maintenance \[page 42\]](#)

2.6.3 Data Modification and Caching

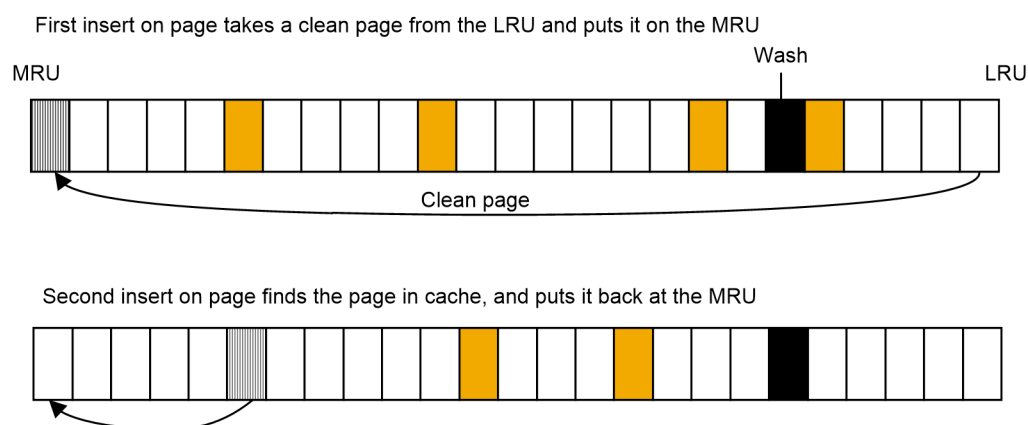
SAP ASE tries to minimize disk writes by keeping changed pages in cache. Many users can make changes to a data page while it resides in the cache. The changes are logged in the transaction log, but the changed data and index pages are not immediately written to disk.

2.6.3.1 Caching and Inserts on Heap Tables

Inserts to heap tables take place on the last page of a table that uses allpages locking and on a page that was recently used for a successful insert, on a table that uses data-only-locking.

If an insert is the first row on a new page for the table, a clean data buffer is allocated to store the data page. This page starts to move down the MRU/LRU chain in the data cache as other processes read pages into memory.

If a second insert to the page takes place while the page is still in memory, the page is located in cache, and moves back to the top of the MRU/LRU chain.



The changed data page remains in cache until it reaches the LRU end of the chain of pages. The page may be changed or referenced many times while it is in the cache, but it is written to disk only when one of the following takes place:

- The page moves past the wash marker.
- A checkpoint or the housekeeper wash task writes it to disk.

See Chapter 5, “Memory Use and Performance” in *Performance and Tuning Series: Basics*.

2.6.3.2 Caching, update, and delete Operations on Heap Tables

When you update or delete a row from a heap table, the effects on the data cache are similar to the process for inserts. If a page is already in the cache, the row is changed and the entire buffer (a single page or more, depending on the I/O size) is placed on the MRU end of the chain.

If the page is not in cache, it is read from disk into cache and examined to determine whether the rows on the page match query clauses. Its placement on the MRU/LRU chain depends on whether data on the page needs to be changed:

- If data on the page needs to be changed, the buffer is placed on the MRU end. It remains in cache, where it can be updated repeatedly, or read by other users before being flushed to disk.
- If data on the page does not need to be changed, the buffer is placed immediately before the wash marker in the cache.

3 Setting Space Management Properties

Setting space management properties can help reduce the amount of maintenance work required to maintain high performance for tables and indexes

3.1 Index Maintenance Reduction

The `fillfactor` option for the `create index` command allows you to specify how full to make index pages and the data pages of clustered indexes.

When you specify a `fillfactor` value of any amount other than 100%, data and index rows use more disk space than the default setting requires.

If you are creating indexes for tables that will grow in size, you can reduce the impact of page splitting on your tables and indexes by using the `fillfactor` option.

`fillfactor` is used when you create an index, and again when you use `reorg rebuild` to rebuild indexes as part of table reorganization operations (for example, when you rebuild clustered indexes or run `reorg rebuild` on a table). `fillfactor` values are not saved in `sysindexes`, and the fullness of the data or index pages is not maintained over time. `fillfactor` is not maintained over time during subsequent inserts or updates to the table.

If the leaf-level pages of your index are initially only partially full (because of the `fillfactor` value), but this free space is used because of subsequent insertions, the leaf-level pages are prone to future splits. Use `reorg rebuild...index` to build the leaf-level pages, creating them with the specified value for `fillfactor` so that future insertions do not cause these splits. Run `reorg rebuild` on the entire index level so the value for `fillfactor` allows additional space at the leaf level for the whole index. If there is a local index, run `reorg rebuild index` at the partition level so only leaf pages in the local index partition are adjusted, leaving additional space for future inserts at the leaf level.

i Note

SAP ASE 15.0 and later allows you to run `reorg rebuild...index` on local index partitions.

When you issue `create index`, 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.

You can also use `sp_chgattribute` to store values for `fillfactor` that are used when `reorg rebuild` is run on a table.

Related Information

[Set fillfactor Values \[page 51\]](#)

3.1.1 Advantages of Using fillfactor

Setting `fillfactor` to a low value provides a temporary performance enhancement. As inserts to the database increase the amount of space used on data or index pages, its performance improvement decreases.

Using a lower value for `fillfactor`:

- Reduces page splits on the leaf-level of indexes, and the data pages of allpages-locked tables.
- Improves data-row clustering on data-only-locked tables with clustered indexes that experience inserts.
- Can reduce lock contention for tables that use page-level locking, since it reduces the likelihood that two processes will need the same data or index page simultaneously.
- Can help maintain large I/O efficiency for the data pages and for the leaf levels of nonclustered indexes, since page splits occur less frequently. This means that eight pages on an extent are likely to be sequential.

3.1.2 Disadvantages of Using fillfactor

If you use `fillfactor` (especially with a very low value), you may notice some disadvantageous effects on queries and maintenance activities.

- More pages must be read for each query that performs a table scan or leaf-level scan on a nonclustered index.
In some cases, a level may be added to an index's B-tree structure, since there will be more pages at the data level and possibly more pages at each index level.
- Increased index size, reducing the index's space efficiency. Because you cannot tune the `fillfactor` value at the page level, page splits with skewed data distribution occur frequently, even when there is available reserved space.
- `dbcc` commands take more time because they must check more pages.
- The time required to run `dump database` increases because more pages must be dumped. `dump database` copies all pages that store data, but does not dump pages that are not yet in use. Dumps and loads may also use more tapes.
- `Fillfactor` values fade over time. If you use `fillfactor` to reduce the performance impact of page splits, monitor your system and recreate indexes when page splitting begins to hurt performance.

3.1.3 Set fillfactor Values

Use `sp_chgattribute` 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` against tables using any locking scheme.
- Use `alter table...partition by` to repartition a table.
- Use `alter table...lock` to change the locking scheme for a table. or use an `alter table...add/modify` command that requires copying the table.
- Run `create clustered index` and a value is stored for the table.

See the *Reference Manual: Commands* for details information about each of these commands.

With the default fillfactor of 0, the index management process leaves room for two additional rows on each index page when you create a new index. When you set fillfactor to 100 percent, it no longer leaves room for these rows. The only effect that `fillfactor` has on size calculations is when calculating the number of clustered index pages and when calculating the number of non-leaf pages. Both of these calculations subtract 2 from the number of rows per page. Eliminate the -2 from these calculations.

Other values for `fillfactor` reduce the number of rows per page on data pages and leaf index pages. To compute the correct values when using `fillfactor`, multiply the size of the available data page (2016) by the `fillfactor`. For example, if your `fillfactor` is 75 percent, your data page would hold 1471 bytes. Use this value in place of 2016 when you calculate the number of rows per page.

SAP ASE does not apply the stored `fillfactor` when it builds nonclustered indexes as a result of a `create clustered index` command:

- 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 fillfactor percent` configuration parameter) is applied to all indexes.

Related Information

[Compute the Number of Data Pages \[page 80\]](#)

[Calculate the Number of Leaf Pages in the Index \[page 85\]](#)

3.1.4 fillfactor Examples

Examples demonstrating the application of `fillfactor` values.

3.1.4.1 No Stored fillfactor Values

With no `fillfactor` values stored in `sysindexes`, SAP ASE applies the `fillfactor` specified in `create index`.

For example:

```
create clustered index title_id_ix
on titles (title_id)
```

```
with fillfactor = 80
```

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 `create clustered index`, the nonclustered indexes always use the server-wide default; they never use a value from `sysindexes`.

3.1.4.1.1 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 `default fillfactor percentage`.

The default `fillfactor` is applied as shown:

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

3.1.4.2 Table-Level or Clustered Index fillfactor Value Stored

An example of stored `fillfactor` value for a table and how the `create clustered index` command applies the the `fillfactor`.

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

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

If you set the stored table-level value for `fillfactor` to 50, this `create clustered index` command applies the `fillfactor`.

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

Table 3: Using Stored fillfactor Values for Clustered Indexes

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

Note

When you run `create clustered index`, any table-level `fillfactor` value stored in `sysindexes` is reset to 0.

You must first issue `sp_chgattribute` to specify that data-only-locked data pages are filled during a `create clustered index` or `reorg` command.

3.1.4.2.1 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.

3.1.4.2.2 Tables with Clustered Indexes

You can set the `fillfactor` value for the data pages by providing either the table name or the clustered index name.

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. 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 you change a DOL table to use allpages locking, the `fillfactor` stored for the table is used for the data pages. SAP ASE ignores the `fillfactor` stored for the clustered index.

This table shows the `fillfactor` values that are set on data and index pages using an `alter table...lock` command, executed after the `sp_chgattribute` commands above have been run.

<code>alter table...lock</code>	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.

3.1.4.2.3 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
```

This table 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.

<code>reorg rebuild</code>	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

3.1.5 sorted_data and fillfactor Option Usage

Use the `sorted_data` option for `create index` when the data to be sorted is already in an order specified by the index key.

This allows `create clustered index` to skip data sorting, reallocating, and rebuilding the table's data pages.

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. However, the use of other `create index` options might still require copying.

Related Information

[Create an Index on Sorted Data \[page 100\]](#)

3.2 Row Forwarding Reduction

You may want to specify an expected row size for a data-only-locked table when an application allows rows with null values or short variable-length character fields to be inserted, and these rows grow in length with subsequent updates.

Set an expected row size to reduce row forwarding.

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 you can 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 may result in row forwarding.

Set the expected row size for tables with variable-length columns, using:

- `exp_row_size` parameter, in a `create table` statement.
- `sp_chgattribute`, for an existing table.
- 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 you later convert the table to data-only locking, SAP ASE applies the `exp_row_size` during the conversion process and to all subsequent inserts and updates. The value for `exp_row_size` applies to the entire table.

Related Information

[Data-Only-Locked Heap Tables \[page 41\]](#)

3.2.1 Default, Minimum, and Maximum Values for `exp_row_size`

Minimum and maximum values for expected row size and the meaning of the special values 0 and 1.

<code>exp_row_size</code> Values	Minimum, Maximum, and Special Values
Minimum	The greater of: <ul style="list-style-type: none">• 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, since they are zero-length when null.

3.2.1.1 Default Value

If you do not specify an expected row size or a value of 0 when you create a data-only-locked table with variable-length columns, SAP ASE uses the amount of space specified by the configuration parameter `default exp_row_size percent` for any table that has variable-length columns.

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

Related Information

[Set a Default Expected Row Size Server-Wide \[page 58\]](#)

3.2.2 Specify an Expected Row Size with Create Table

An example demonstrating how `create table` specifies an expected row size of 200 bytes.

```
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

```

3.2.3 Add or Change an Expected Row Size

Use `sp_chgattribute` to add or change the expected row size for a table.

For example, to set the expected row size to 190 for the `new_titles` table, enter:

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

To switch the row size for a table from a current, explicit value to the default `exp_row_size percent`, enter:

```
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 you create a clustered index on the table or run `reorg rebuild`. The expected row size is applied as rows are copied to new data pages.
If you increase `exp_row_size`, and recreate 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.

3.2.4 Set a Default Expected Row Size Server-Wide

`default exp_row_size percent` reserves a percentage of the page size to set aside for expanding updates.

The default value, 5, sets aside 5% 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 sets aside 100 bytes for row expansion. This command sets the default value to 10%:

```
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.

3.2.5 Display 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_row_size` column for all user tables in a database:

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

3.2.6 Choose 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 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 space on data pages due to over-allocating space towards the expected row size value.

3.2.6.1 Use `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
```

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.

See Chapter 2, "Statistics Tables and Displaying Statistics with `optdiag`," in the *Performance and Tuning Series: Improving Performance with Statistical Analysis*.

3.2.6.2 Query systabstats for Forwarded Rows

The `forwrowcnt` column in the `systabstats` table stores the number of forwarded rows for a table.

To display the number of forwarded rows and average row size for all user tables with object IDs greater than 100, use this query:

```
select objectname = object_name(id),
       partitionname = (select name from syspartitions p
                        where p.id = t.id and p.indid = t.indid)
       , forwrowcnt, datarowsize
       , exprowsize = (select i.exp_rowsize from sysindexes i
                       where i.id = t.id and i.indid = t.indid)
into #temptable
from systabstats t
where id > 100 and indid IN (0,1)
exec sp_autoformat #temptable
```

i Note

Forwarded row counts are updated in memory, and the housekeeper tasks periodically flushes them to disk.

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.

3.2.7 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.

Value of <code>max_rows_per_page</code>	Value of <code>exp_row_size</code>
0	Percentage value set by default <code>exp_row_size percent</code>
1 – 254	The smaller of: <ul style="list-style-type: none">• Maximum row size• (logical page size) – (page header overheads) / <code>max_rows_per_page</code>

For example, if `max_rows_per_page` is set to 10 for an allpages-locked table on a server configured for 2K pages with a maximum defined row size of 300 bytes, the `exp_row_size` value is 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 is set to 150.

3.2.8 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 determine the number of forwarded rows being generated by your applications.

Run `reorg forwarded_rows` the number of forwarded rows is high enough to affect application performance. `reorg forwarded_rows` uses short transactions and is nonintrusive, so you can run it while applications are active.

See "Using the `reorg` Command," in the *System Administration Guide: Volume 2*.

You can monitor forwarded rows on a per-partition basis, and run `reorg forwarded_rows` on those partitions that have a large number of forwarded rows. See the *Reference Manual: Commands*.

If the application continues to generate a large number of forwarded rows, consider using `sp_chgattribute` to increase the expected row size for the table.

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 nonpeak 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 required to read a set of rows. If the increase in the number of I/Os due to increased storage space is high, allowing rows to be forwarded and occasionally running `reorg` may have less overall performance impact.

3.3 Space for Forwarded Rows and Inserts

Set a `reservepagegap` value to reduce storage fragmentation, thus also reducing the frequency of maintenance activities such as running `reorg rebuild` and recreating indexes for some tables.

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. Use the `reservepagegap` space management property 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 you create a clustered index on a table or immediately after you run `reorg rebuild`. However, future data modifications may cause row forwarding and require allocation of additional data and index pages to store inserted rows.

You can set the reserve page gap on the data and index layer pages for allpages and data-only-locked tables.

3.3.1 Extent Allocation Commands and reservepagegap

Extent allocation means that pages are allocated in multiples of eight, rather than one page at a time. This reduces logging activity by writing only 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. `alter table` commands that add, drop, or modify columns, or change a table's partitioning scheme sometimes also require a table-copy operation. By default, all these commands use extent allocation.

Specify `reservepagegap` value in pages, indicating a ratio of empty pages to filled pages. For example, if you specify a `reservepagegap` value of 8, an operation that uses extent allocation fills seven pages and leaves the eighth page empty.

Extent allocation operations 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 seven empty, unallocated pages, 248 used pages, and the allocation page. SAP ASE can use the seven empty pages 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.

See Chapter 12, "Creating Indexes on Tables" in the *Transact-SQL Users Guide* for information about when to use `reservepagegap`.

This table 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.

256	257	258	259	260	261	262	263
264	265	266	267	268	269	270	271
272	273	274	275	276	277	278	279
280	281	282	283	284	285	286	287
288	289	290	291	292	293	294	295
296	297	298	299	300	301	302	303
304	305	306	306	308	309	310	311
...							
504	505	506	507	508	509	510	511

- Allocation page
- Pages used by the object
- Reserved pages
- Unallocated pages

3.3.2 Specify a Reserve Page Gap with create table

Use the `reservepagegap` command with `create table` to specify a reserve page gap.

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
```

Any operation that performs extent allocation on the `more_titles` table leaves 1 empty page for each 15 filled pages. For partitioned tables, the `reservepagegap` value applies to all partitions.

The default value for `reservepagegap` is 0, meaning that no space is reserved.

3.3.3 Specify a Reserve Page Gap with create index

Use the `reservepagegap` command with `create index` to specify a reserve page gap.

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

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

You can specify a `reservepagegap` value with the `alter table...constraint` options, `primary key` and `unique`, that create indexes. The value of `reservepagegap` for local index on partitioned tables applies to all local index partitions.

This example creates a unique constraint:

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

3.3.4 Change reservepagegap

`sp_chgattribute` changes only 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.

It changes 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.
- Any command that copies the table's data to create a new version of the table applies the reserve page gap during the data copy phase of the operation. For example, using `reorg rebuild` or using `alter table` to change the locking or partitioning scheme of a table or any change of schema that requires a data copy both apply to reserve page gap.
- When you create a clustered index, the reserve page gap value stored for the table is applied to the data pages.

To change the reserve page gap for the `titles` table to 20, enter:

```
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 reserve page gap is applied to index pages during:

- `alter table...lock`, indexes are rebuilt.
- The index rebuild phase during `reorg rebuild` when using `alter table` to change the locking or partitioning scheme of a table, or when changing any schema that requires a data copy.
- `create clustered index` and `alter table` commands that create a clustered index, as nonclustered indexes are rebuilt

3.3.5 reservepagegap Examples

Examples showing how `reservepagegap` is applied during `alter table` and `reorg rebuild` commands.

3.3.5.1 reservepagegap Specified Only for the Table

An example using the `reservepagegap` command for the table, without specifying a value in the `create index` commands.

For example:

```
sp_chgattribute titles, "reservepagegap", 16
```

```
create clustered index title_ix on titles(title_id)
```

```
create index type_price on titles(type, price)
```

This table shows the values applied when running `reorg rebuild` or dropping and creating a clustered index.

Command	Allpages-Locked Table	Data-Only-Locked Table
<code>create clustered index or clustered index rebuild due to reorg rebuild</code>	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)

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

3.3.5.2 `reservepagegap` Specified for a Clustered Index

An example demonstrating different `reservepagegap` values for the table and the clustered index, and a value for the nonclustered `type_price` index.

For example:

```
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
```

This table shows the effects of this sequence of commands.

Command	Allpages-Locked Table	Data-Only-locked Table
<code>create clustered index or clustered index rebuild due to reorg rebuild</code>	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.

3.3.6 Choose a Value for `reservepagegap`

Certain factors depend on the value you choose 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
- The frequency with which you recreate 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.

3.3.7 Monitor reservepagegap Settings

Use `optdiag` to check the cluster ratio and the number of forwarded rows in tables. Declines in cluster ratios may also indicate that you can improve performance by running `reorg` commands.

- If the data page cluster ratio for a clustered index is low, run `reorg rebuild` or drop and recreate the clustered index.
- If the index page cluster ratio is low, drop and recreate the nonclustered index.

To reduce the frequency with which you run `reorg` commands to maintain cluster ratios, increase the `reservepagegap` slightly before running `reorg rebuild`.

See Chapter 2, “Statistics Tables and Displaying Statistics with `optdiag`,” in *Performance and Tuning Series: Improving Performance with Statistical Analysis*.

3.3.8 reservepagegap and sorted_data Options

`sorted_data` suppresses copying the data pages in key order for unpartitioned tables when a clustered index is created on a table that is stored on the data pages in index key order. Use `reservepagegap` in `create clustered index` commands to leave empty pages on the extents used by the table.

There are rules that determine which option takes effect. You cannot use `sp_chgattribute` to change the `reservepagegap` value and get the benefits of both of these options.

If you specify both 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 `create clustered index` applies only to the index pages. Data pages are not copied.

Besides `reservepagegap`, other options to `create clustered index` may require a sort, which causes the `sorted_data` option to be ignored.

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.

These tables shows how these rules apply.

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

3.3.8.1 Redistribute the Data pages of a Table

Options to use for allpages-locked tables and data-only-locked tables to redistribute the data pages of a table, leaving room for later expansion.

- For allpages-locked tables, drop and recreate the clustered index without using the `sorted_data` option. If the value stored in `sysindexes` is not the value you want, use `create clustered index` to specify the desired `reservepagegap`.
- For data-only-locked tables, use `sp_chgattribute` to set the `reservepagegap` for the table to the desired value, then drop and recreate 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.

3.3.8.2 Create a Clustered Index Without Copying Data Pages

Options to use for allpages-locked tables and data-only-locked tables to create a clustered index without copying data pages.

- For allpages-locked tables, use the `sorted_data` option, but do not use `create clustered index` to specify a `reservepagegap`. Alternatively, 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.

3.3.8.3 `sorted_data` Option Following an Extent Allocation Command

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
```

3.4 `max_rows_per_page` Usage on allpages-locked Tables

Setting a maximum number of rows per pages can reduce contention for allpages-locked tables and indexes.

In most cases, it is preferable to convert the tables to use a data-only-locking scheme. If there is some reason that you cannot change the locking scheme, and contention is a problem on an allpages-locked table or index, setting a `max_rows_per_page` value may help performance.

When there are fewer rows on the index and data pages, the chances of lock contention are reduced. As the keys are spread out over more pages, it becomes more likely that the page you want is not the page someone else needs. To change the number of rows per page, adjust the `fillfactor` or `max_rows_per_page` values of your tables and indexes.

`fillfactor` (defined by either `sp_configure` or `create index`) determines how full SAP ASE makes each data page when it creates a new index on existing data. Since `fillfactor` helps reduce page splits, exclusive locks are also minimized on the index, improving performance. However, the `fillfactor` value is not maintained by subsequent changes to the data. `max_rows_per_page` (defined by `sp_chgattribute`, `create index`, `create table`, or `alter table`) is similar to `fillfactor`, except that SAP ASE maintains the `max_rows_per_page` value as the data changes.

The costs associated with decreasing the number of rows per page using `fillfactor` or `max_rows_per_page` include more I/O to read the same number of data pages, more memory for the same performance from the data cache, and more locks. In addition, a low value for `max_rows_per_page` for a table may increase page splits when data is inserted into the table.

3.4.1 Lock Contention Reduction

The `max_rows_per_page` value specified in a `create table`, `create index`, or `alter table` command restricts the number of rows allowed on a data page, a clustered index leaf page, or a nonclustered index leaf page. This reduces lock contention and improves concurrency for frequently accessed tables.

The `max_rows_per_page` value specified in a `create table`, `create index`, or `alter table` command restricts the number of rows allowed on a data page, a clustered index leaf page, or a nonclustered index leaf page. This reduces lock contention and improves concurrency for frequently accessed tables.

`max_rows_per_page` applies to the data pages of a heap table, or the leaf pages of an index. Unlike `fillfactor`, which is not maintained after creating a table or index, SAP ASE retains the `max_rows_per_page` value when adding or deleting rows.

The following command creates the `sales` table and limits the maximum rows per page to four:

```
create table sales
  (stor_id          char(4)          not null,
   ord_num         varchar(20)      not null,
   date            datetime         not null)
with max_rows_per_page = 4
```

If you create a table with a `max_rows_per_page` value, and then create a clustered index on the table without specifying `max_rows_per_page`, the clustered index inherits the `max_rows_per_page` value from the `create table` statement. Creating a clustered index with `max_rows_per_page` changes the value for the table's data pages.

3.4.2 Indexes and max_rows_per_page

The default value for `max_rows_per_page` is 0, which creates clustered indexes with full data pages, creates nonclustered indexes with full leaf pages, and leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes.

For heap tables and clustered indexes, the range for `max_rows_per_page` is 0 – 256.

For nonclustered indexes, the maximum value for `max_rows_per_page` is the number of index rows that fit on the leaf page, without exceeding 256. To determine the maximum value, subtract 32 (the size of the page header) from the page size and divide the difference by the index key size. The following statement calculates the maximum value of `max_rows_per_page` for a nonclustered index:

```
select (@@pagesize - 32)/minlen
  from sysindexes
  where name = "<indexname>"
```

3.4.3 select into and max_rows_per_page

By default, `select into` does not carry over a base table's `max_rows_per_page` value, but creates the new table with a `max_rows_per_page` value of 0. However, you can add the `with max_rows_per_page` option to `select into` to specify a value other than 0.

3.4.4 Applying max_rows_per_page to Existing Data

There are several ways to apply a `max_rows_per_page` value to existing data.

Context

- If the table has a clustered index, drop and recreate the index using a different `max_rows_per_page` value.
- Use `sp_chgattribute` to change the value of `max_rows_per_page`, then rebuild the entire table and its indexes with `reorg rebuild`. For example, to change the `max_rows_per_page` value of the `authors` table to 1, enter:

```
sp_chgattribute authors, "max_rows_per_page", 1
go
reorg rebuild authors
go
```

- Use `bcpl` to repopulate the table, and:
 1. Copy out the table data.
 2. Truncate the table.
 3. Use `sp_chgattribute` to set the `max_rows_per_page` value.
 4. Copy the data back in.

4 Table and Index Size

This chapter explains how to determine the current sizes of tables and indexes and how to estimate table size for space planning.

Knowing the sizes of your tables and indexes is important to understanding query and system behavior. At several stages of tuning work, you need size data to:

- Understand `statistics io` reports for a specific query plan. Chapter 1, “Using the set statistics Commands,” in *Performance and Tuning Series: Improving Performance with Statistical Analysis* describes how to use `statistics io` to examine the I/O performed.
- Understand the optimizer’s choice of query plan. The SAP ASE cost-based optimizer estimates the physical and logical I/O required for each possible access method and chooses the cheapest method. If you think a particular query plan is unusual, use `dbcc traceon(302)` to determine why the optimizer made the decision. This output includes page number estimates.
- Determine object placement, based on the sizes of database objects and the expected I/O patterns on the objects. Improve performance by distributing database objects across physical devices so that reads and writes to disk are evenly distributed.
- Understand changes in performance. If objects grow, their performance characteristics can change. One example is a table that is heavily used and is usually 100% cached. If that table grows too large for its cache, queries that access the table can suddenly suffer poor performance. This is particularly true for joins requiring multiple scans.
- Perform capacity planning. Whether you are designing a new system or planning for growth of an existing system, you must know your space requirements to plan for physical disks and memory needs.
- Understand output from SAP ASE Monitor and from `sp_sysmon` reports on physical I/O.

4.1 Determine the Sizes of Tables and Indexes

SAP ASE includes several tools that provide information about the current sizes of tables or indexes, or that can predict future sizes.

- `optdiag` displays the sizes and many other statistics for tables and indexes. See Chapter 2, “Statistics Tables and Displaying Statistics with `optdiag`,” in *Performance and Tuning Series: Improving Performance with Statistical Analysis*
- `sp_spaceused` reports on the current size of existing tables and indexes.
- `sp_estspace` can predict the size of a table and its indexes, given a number of rows as a parameter.

You can also compute table and index size using formulas provided in this chapter. `sp_spaceused` and `optdiag` report actual space usage. The other methods presented in this chapter provide size estimates.

For partitioned tables, `sp_helppartition` reports on the number of pages stored on each partition of the table. See Chapter 10, “Partitioning Tables and Indexes” in the *Transact-SQL Users Guide*.

4.2 Effects of Data Modifications on Object Sizes

Over time, the effects of randomly distributed data modifications on a set of tables tend to produce data pages and index pages that average approximately 75% full.

The major factors are:

- When you insert a row to be placed on a page of an allpages-locked table with a clustered index, and there is no room on the page for that row, the page is split, leaving two pages that are about 50 percent full.
- When you delete rows from heaps or from tables with clustered indexes, the space used on the page decreases. You can have pages that contain very few rows or even a single row.
- After some deletes or page splits have occurred, inserting rows into tables with clustered indexes tends to fill up pages that have been split, or pages where rows have been deleted.

Page splits also take place when rows need to be inserted into full index pages, so index pages also tend to average approximately 75% full, unless you drop and recreate them periodically.

4.3 optdiag Usage to Display Object Sizes

The `optdiag` command displays statistics for tables, indexes, and columns, including the size of tables and indexes.

If you are performing query tuning, `optdiag` provides the best tool for viewing all the statistics you need. Here is a sample report for the `titles` table in the `pubtune` database:

```
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
```

See Chapter 2, “Statistics Tables and Displaying Statistics with `optdiag`,” in *Performance and Tuning Series: Improving Performance with Statistical Analysis*.

4.3.1 Advantages of `optdiag`

There are several advantages of `optdiag`.

The advantages are:

- It can display statistics for all tables in a database, or for a single table.
- `optdiag` output contains additional information useful for understanding query costs, such as index height and the average row length.

- It is frequently used for other tuning tasks, so you may have these reports readily available.

4.3.2 Disadvantages of `optdiag`

The principle disadvantage of `optdiag` is that it produces a lot of output. If you need only a single piece of information, such as the number of pages in a table, other methods are faster and incur lower system overhead.

4.4 `sp_spaceused` Usage to Display Object Size

The system procedure `sp_spaceused` reads values stored on an object's OAM page to provide a quick report on the space used by the object.

The syntax is:

```
sp_spaceused titles

name          rowtotal reserved      data          index_size    unused
-----
titles        5000          1756 KB      1242 KB       440 KB        74 KB
```

The `rowtotal` value may be inaccurate at times; not all SAP ASE processes update this value on the OAM page. The commands `update statistics`, `dbcc checktable`, and `dbcc checkdb` correct the `rowtotal` value on the OAM page.

This table explains the headings in `sp_spaceused` output.

Column	Meaning
<rowtotal>	Reports an estimate of the number of rows. The value is read from the OAM page. Though not always exact, this estimate is much quicker and leads to less contention than <code>select count (*)</code> .
<reserved>	Reports pages reserved for use by the table and its indexes. It includes both the used and unused pages in extents allocated to the objects. It is the sum of <code>data</code> , <code>index_size</code> , and <code>unused</code> .
<data>	Reports the kilobytes on pages used by the table.
<index_size>	Reports the total kilobytes on pages used by the indexes.
<unused>	Reports the kilobytes of unused pages in extents allocated to the object, including the unused pages for the object's indexes.

To report index sizes separately, use:

```
sp_spaceused titles, 1
```

index_name	size	reserved	unused
title_id_cix	14 KB	1294 KB	38 KB
title_ix	256 KB	272 KB	16 KB
type_price_ix	170 KB	190 KB	20 KB

name	rowtotal	reserved	data	index_size	unused
titles	5000	1756 KB	1242 KB	440 KB	74 KB

For clustered indexes on allpages-locked tables, the `size` value represents the space used for the root and intermediate index pages. The `reserved` value includes the index size and the reserved and used data pages.

The “1” in the `sp_spaceused` syntax indicates that detailed index information should be printed. It has no relation to index IDs or other information.

4.4.1 Advantages of `sp_spaceused`

There are several advantages of `sp_spaceused`.

That advantages are that:

- It provides quick reports without excessive I/O and locking, since it uses only values in the table and index OAM pages to return results.
- It shows the amount of space that is reserved for expansion of the object, but not currently used to store data.
- It provides detailed reports on the size of indexes and of `text` and `image`, and Java off-row column storage.

i Note

Use `sp_helppartition` to report the number of pages in each partition. `sp_helppartition` does not report the same level of detail as `sp_spaceused`, but does give a general idea of the amount of space a partition uses. In SAP ASE version 15.0.2 and later, `sp_spaceusage` provides detailed information about a variety of subjects, including the space used by tables at the index and partition level, and fragmentation.

See the *SAP ASE Reference Manual: Procedures* for more information about all these system procedures.

4.4.2 Disadvantages of `sp_spaceused`

There are a few disadvantages of `sp_spaceused`.

The disadvantages are that:

- It may report inaccurate counts for row total and space usage.
- Output is only in kilobytes, while most query-tuning activities use pages as a unit of measure. However, you can use `sp_spaceusage` to report information in any unit you specify.

4.5 Using sp_estspace to Estimate Object Size

sp_spaceused and optdiag report on actual space usage. sp_estspace can help you plan for future growth of your tables and indexes.

Context

This procedure uses information in the system tables (sysobjects, syscolumns, and sysindexes) to determine the length of data and index rows. You provide a table name, and the number of rows you expect to have in the table, and sp_estspace estimates the size for the table and for any indexes that exist. It does not look at the actual size of the data in the tables.

To use sp_estspace:

- Create the table, if it does not already exist.
- Create any indexes on the table.
- Execute the procedure, estimating the number of rows that the table will hold.

The output reports the number of pages and bytes for the table and for each level of the index.

The following example estimates the size of the titles table with 500,000 rows, a clustered index, and two nonclustered indexes:

```
sp_estspace titles, 500000
```

name	type	idx_level	Pages	Kbytes
titles	data	0	50002	100004
title_id_cix	clustered	0	302	604
title_id_cix	clustered	1	3	6
title_id_cix	clustered	2	1	2
title_ix	nonclustered	0	13890	27780
title_ix	nonclustered	1	410	819
title_ix	nonclustered	2	13	26
title_ix	nonclustered	3	1	2
type_price_ix	nonclustered	0	6099	12197
type_price_ix	nonclustered	1	88	176
type_price_ix	nonclustered	2	2	5
type_price_ix	nonclustered	3	1	2
Total_Mbytes				

138.30				
name	type	total_pages	time_mins	

title_id_cix	clustered	50308	250	
title_ix	nonclustered	14314	91	
type_price_ix	nonclustered	6190	55	

sp_estspace also allows you to specify a fillfactor, the average size of variable-length fields and text fields, and the I/O speed. For more information, see *Reference Manual: Procedures*.

i Note

The index creation times printed by sp_estspace do not factor in the effects of parallel sorting.

4.5.1 Advantages of `sp_estspace`

There are a few advantages of `sp_estspace`.

The advantages are that it:

- Provides an efficient way to perform initial capacity planning and to plan for table and index growth.
- Helps you estimate the number of index levels.
- Helps you estimate future disk space, cache space, and memory requirements.

4.5.2 Disadvantages of `sp_estspace`

There are a few disadvantages of `sp_estspace`.

The disadvantages are that:

- Returned sizes are only estimates and may differ from actual sizes, due to fillfactors, page splitting, actual size of variable-length fields, and other factors.
- Index creation times can vary widely, depending on disk speed, the use of extent I/O buffers, and system load.

4.6 Use Formulas to Estimate Object Size

Formulas to help you estimate the future sizes of the tables and indexes in your database.

The amount of overhead in each row for tables and indexes that contain variable-length fields is greater than the overhead for tables that contain only fixed-length fields, so two sets of formulas are required.

The process involves calculating the number of bytes of data and overhead for each row, and dividing that number into the number of bytes available on a data page. Each page requires some overhead, which limits the number of bytes available for data:

- For allpages-locked tables, page overhead is 32 bytes, leaving 2016 bytes available for data on a 2K page.
- For data-only-locked tables, page overhead is 46 bytes, leaving 2002 bytes available for data.

For the most accurate estimate, round down divisions that calculate the number of rows per page (rows are never split across pages), and round up divisions that calculate the number of pages.

4.6.1 Factors that can Affect Storage Size

Using space management properties can increase the space needed for a table or an index.

If your table includes `text` or `image` datatypes or Java off-row columns, use 16 (the size of the text pointer that is stored in the row) in your calculations. Then refer to LOB pages to see how to calculate the storage space required for the actual `text` or `image` data.

Indexes on data-only-locked tables may be smaller than the formulas predict due to two factors:

- Duplicate keys are stored only once, followed by a list of row IDs for the key.
- Compression of keys on nonleaf levels; only enough of the key to differentiate from the neighboring keys is stored. This is especially effective in reducing the size when long character keys are used.

If the configuration parameter `page_utilization_percent` is set to less than 100, SAP ASE may allocate new extents before filling all pages on the allocated extents. This does not change the number of pages used by an object, but leaves empty pages in the extents allocated to the object.

Related Information

[Effects of Space Management Properties \[page 94\]](#)

[max_rows_per_page \[page 96\]](#)

[LOB Pages \[page 96\]](#)

4.6.2 Storage Sizes for datatypes

Storage sizes for SAP ASE datatypes.

Datatype	Size
char	Defined size
nchar	Defined size * <code><@@ncharsize></code>
unichar	<code>n*@@unicharsize<></code> (<code>@@unicharsize equals 2</code>)>
univarchar	the actual number of characters* <code><@@unicharsize></code>
varchar	Actual number of characters
nvarchar	Actual number of characters * <code><@@ncharsize></code>
binary	Defined size
varbinary	Data size
int	4
smallint	2
tinyint	1
float	4 or 8, depending on precision

Datatype	Size
double precision	8
real	4
numeric	2–17, depending on precision and scale
decimal	2–17, depending on precision and scale
money	8
smallmoney	4
datetime	8
smalldatetime	4
bit	1
text	16 bytes + 2K * number of pages used
image	16 bytes + 2K * number of pages used
timestamp	8

The storage size for a `numeric` or `decimal` column depends on its precision. The minimum storage requirement is 2 bytes for a 1- or 2-digit column. Storage size increases by 1 byte for each additional 2 digits of precision, up to a maximum of 17 bytes.

Any columns defined as `NULL` are considered variable-length columns, since they involve the overhead associated with variable-length columns.

All calculations in the examples that follow are based on the maximum size for `varchar`, `univarchar`, `nvarchar`, and `varbinary` data—the defined size of the columns. They also assume that the columns were defined as `NOT NULL`.

4.6.3 Tables and Indexes Used in the Formulas

Examples illustrate the computations on a table that contains 9,000,000 rows.

- The sum of fixed-length column sizes is 100 bytes.
- The sum of variable-length column sizes is 50 bytes; there are 2 variable-length columns.

The table has two indexes:

- A clustered index, on a fixed-length column, of 4 bytes
- A composite nonclustered index with these columns:
 - A fixed length column, of 4 bytes
 - A variable length column, of 20 bytes

Different formulas are needed for allpages-locked and data-only-locked tables, since they have different amounts of overhead on the page and per row.

Related Information

[Calculating Table and Clustered Index Sizes for allpages-locked Tables \[page 79\]](#)

[Calculating the Sizes of Data-Only-Locked Tables \[page 88\]](#)

4.6.4 Calculating Table and Clustered Index Sizes for allpages-locked Tables

Formulas to calculate the sizes of tables and clustered indexes.

Context

The formulas and examples for allpages-locked tables are listed below as a series of steps. Steps 1–6 outline the calculations for an allpages-locked table with a clustered index, giving the table size and the size of the index tree. Steps 7–12 outline the calculations for computing the space required by nonclustered indexes. All of the formulas use the maximum size of the variable-length fields.

Procedure

1. Calculate the data row size.
2. Compute the number of data pages.
3. Compute the size of clustered index rows.
4. Compute the number of clustered index pages.
5. Compute the total number of index pages.
6. Calculate allocation overhead and total pages.
7. Calculate the size of the leaf index row
8. Calculate the number of leaf pages in the index.
9. Calculate the size of the nonleaf rows.
10. Calculate the number of non-leaf pages.
11. Calculate the total number of non-leaf index pages.
12. Calculate allocation overhead and total pages.

Results

If your table does not have clustered indexes, skip steps 3, 4, and 5. When you have computed the number of data pages in step 2, go to step 6 to add the number of OAM pages.

`optdiag` output includes the average length of data rows and index rows. You can use these values for the data and index row lengths, if you want to use average lengths instead.

Related Information

[Compute the Number of Data Pages \[page 80\]](#)

[Compute the Size of Clustered Index Rows \[page 81\]](#)

[Compute the Number of Clustered Index Pages \[page 81\]](#)

[Compute the Total Number of Index Pages \[page 82\]](#)

[Calculate Allocation Overhead and Total Pages \[page 83\]](#)

[Calculate the Size of the Leaf Index Row \[page 84\]](#)

[Calculate the Number of Leaf Pages in the Index \[page 85\]](#)

[Calculate the Size of the Non-Leaf Rows \[page 85\]](#)

[Calculate the Number of Non-Leaf Pages \[page 85\]](#)

[Calculate the Total Number of Non-Leaf Index Pages \[page 86\]](#)

[Calculate Allocation Overhead and Total Pages \[page 86\]](#)

[Calculate the Data Row Size \[page 87\]](#)

4.6.4.1 Compute the Number of Data Pages

Formulas to use to compute the number of data pages.

The formulas are:

```
2016 / Data row size = Number of data rows per page
```

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

For example:

```
2016 / 160 = 12 data rows per page
```

```
9,000,000 / 12 = 750,000 data pages
```


4.6.4.2 Compute the Size of Clustered Index Rows

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

4.6.4.2.1 Fixed-Length Columns Only

A formula to use if all of the keys are fixed length. The clustered index in the example has only fixed-length keys.

The formula is:

$$\begin{array}{r} 5 \text{ (Overhead)} \\ + \text{ Sum bytes in the fixed-length index keys} \\ \hline = \text{ Clustered row size} \end{array}$$

For example:

$$\begin{array}{r} 5 \\ + 4 \\ \hline 9 \end{array}$$

4.6.4.2.2 Some Variable-Length Columns

A formula to use if the keys include variable-length columns or allow NULL values.

The formula is:

$$\begin{array}{r} 5 \text{ (Overhead)} \\ + \text{ Sum of bytes in all fixed-length index keys} \\ + \text{ Sum of all bytes in all variable-length index keys} \\ \hline = \text{ Subtotal} \\ + \text{ (Subtotal/ 256) + 1 (Overhead)} \\ + \text{ Number of variable-length columns +1} \\ + 2 \text{ (Overhead)} \\ \hline = \text{ Clustered index row size} \end{array}$$

The results of the division (subtotal / 256) are rounded down.

4.6.4.3 Compute the Number of Clustered Index Pages

Formulas to use to compute the number of clustered index pages.

The formulas are:

$$(2016 / \text{ clustered row size}) - 2 = \text{ number of clustered index rows per page}$$

```
number of rows/ number of clustered index rows per page = number of index
pages at next level
```

For example:

```
(2016/ 9) - 2 = 222
```

```
750,000/ 222 = 3379
```

If the result for the "number of index pages at the next level" 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:

```
Number of index pages at last level/ Number of clustered index rows per page =
Number of index pages at next level
```

For example:

```
3379/ 222 = 16 index pages (Level 1)
```

```
16/ 222 = 1 index page (Level 2)1
```

4.6.4.4 Compute the Total Number of Index Pages

A formula to add the number of pages at each level to determine the total number of pages in the index.

The formula is:

```
Index levels    Pages
2
1               +
0               +
                
Total number of index pages
```

For example:

```
Pages    Rows
  1       16
+ 16     3379
+ 3379   750000
-----
 3379
```

4.6.4.5 Calculate Allocation Overhead and Total Pages

Each table and each index on a table has an object allocation map (OAM). 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
```

For example:

```
750,000 / 63,750 = 12
```

```
750,000 / 2000 = 376
```

To calculate the number of OAM pages for the index, use:

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

```
Number of reserved index pages / 2000 = Maximum OAM pages
```

For example:

```
3396 / 63,750 = 1
```

```
3396 / 2000 = 2
```

4.6.4.5.1 Total Pages Needed

Add the number of OAM pages to earlier totals to determine the total number of pages required.

The formula is:

	Minimum	Maximum
Clustered index pages		
OAM pages	+	+
Data pages	+	+
OAM pages	+	+
<hr/>		
Total		

For example:

Minimum	Maximum
3396	3396
+ 1	2
+750000	750000
+ 12	376
<hr/>	
753409	753773

4.6.4.6 Calculate the Size of the Leaf Index Row

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

4.6.4.6.1 Fixed-Length Keys Only

A formula to use if the index contains only fixed-length keys and are defined as NOT NULL.

The formula is:

$$\begin{array}{r} 7 \text{ (Overhead)} \\ + \quad \text{Sum of fixed-length keys} \\ \hline = \text{Size of leaf index row} \end{array}$$

4.6.4.6.2 Some Variable-Length Keys

A formula to use if the index contains any variable-length keys or columns defined as NULL.

The formula is:

$$\begin{array}{r} 9 \text{ (Overhead)} \\ + \quad \text{Sum of length of fixed-length keys} \\ + \quad \text{Sum of length of variable-length keys} \\ + \quad \text{Number of variable-length keys} + 1 \\ \hline = \text{Subtotal} \\ + \quad (\text{Subtotal} / 256) + 1 \text{ (overhead)} \\ \hline = \text{Size of leaf index row} \end{array}$$

For example:

$$\begin{array}{r} 9 \\ + 4 \\ + 20 \\ + 2 \\ \hline 35 \\ + 1 \\ \hline 36 \end{array}$$

4.6.4.7 Calculate the Number of Leaf Pages in the Index

A formula to calculate the number of leaf pages in the index.

The formula is:

$$(2016 / \text{leaf row size}) = \text{No. of leaf index rows per page}$$

$$\text{No. of table rows} / \text{No. of leaf rows per page} = \text{No. of index pages at next level}$$

For example:

$$2016 / 36 = 56$$

$$9,000,000 / 56 = 160,715$$

4.6.4.8 Calculate the Size of the Non-Leaf Rows

A formula to calculate the size of the nonleaf rows.

The formula is:

$$\begin{array}{r} \text{Size of leaf index row} \\ + \quad 4 \text{ (Overhead)} \\ \hline = \text{Size of the non-leaf row} \end{array}$$

For example:

$$\begin{array}{r} 36 \\ + \quad 4 \\ \hline = 40 \end{array}$$

4.6.4.9 Calculate the Number of Non-Leaf Pages

A formula to calculate the number of non-leaf pages.

The formula is:

$$(2016 / \text{Size of non-leaf row}) - 2 = \text{No. of non-leaf index rows per page}$$

For example:

$$(2016 / 40) - 2 = 48$$

If the number of calculated leaf pages 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 index pages at previous level / No. of non-leaf index rows per page = No.
of index pages at next level
```

For example:

```
160715 / 48 = 3349      Index pages, level 1
3349 / 48 = 70         Index pages, level 2
70 / 48 = 2            Index pages, level 3
2 / 48 = 1             Index page, level 4 (root level)
```

4.6.4.10 Calculate the Total Number of Non-Leaf Index Pages

To determine the total number of pages in the index, add the number of pages at each level.

The formula is:

```
Index levels    Pages
4
3               +
2               +
1               +
0               +
-----
Total number of 2K pages
```

For example:

```
Pages    Rows
  1         2
+  2         70
+ 70       3348
+ 3349     160715
+160715   9000000
-----
164137
```

4.6.4.11 Calculate Allocation Overhead and Total Pages

Formulas to calculate the allocation of overhead and total pages.

The formulas are:

```
number of index pages / 63,750 = minimum OAM pages
```

```
number of index pages / 2000 = maximum OAM pages
```

For example:

```
164137 / 63,750 = 3
```

```
164137 / 2000 = 83
```

Total Pages Needed

Add the number of OAM pages to the total when you calculate the number of non-leaf index pages to determine the total number of index pages.

The formulas are:

```
nonclustered index pages + minimum OAM pages = total
```

```
nonclustered index pages + maximum OAM pages = total
```

For example:

```
164137 + 3 = 164140
```

```
164137 + 83 = 164220
```

4.6.4.12 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.

4.6.4.12.1 Fixed-Length Columns Only

A formula to use if the table contains only fixed-length columns, and all are defined as NOT NULL.

The formula is:

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

4.6.4.12.2 Some Variable-Length Columns

A formula to use if the table contains any variable-length columns or columns that allow NULL values.

The formula is:

```
4 (Overhead)
+ Sum of bytes in all fixed-length columns
```

```

+   Sum of all bytes in all variable-length columns
-----
= Subtotal
+   Subtotal/256) + 1 (Overhead)
+   Number of variable-length columns +1
+   2 (Overhead)
-----
= Data row size

```

For example:

```

      4
+   100
+   50
-----
    154
+     1
+     3
+     2
-----
    160

```

4.6.5 Calculating the Sizes of Data-Only-Locked Tables

Formulas and examples that show how to calculate the sizes of tables and indexes.

Context

This example uses the same column sizes and index as the previous example. All of the formulas use the maximum size of the variable-length fields.

The formulas for data-only-locked tables 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.

`optdiag` output includes the average length of data rows and index rows. You can use these values for the data and index row lengths, if you want to use average lengths instead.

Procedure

1. Calculate the data row size.
2. Compute the number of data pages.
3. Calculate allocation overhead and total pages.
4. Calculate the size of the index row.

5. Calculate the number of leaf pages in the index.
6. Calculate the number of non-leaf pages in the index.
7. Calculate the total number of non-leaf index pages.
8. Calculate allocation overhead and total pages.

Related Information

[Calculate the Data Row Size \[page 89\]](#)

[Compute the Number of Data Pages \[page 90\]](#)

[Calculate Allocation Overhead and Total Pages \[page 90\]](#)

[Calculate the Size of the Index Row \[page 91\]](#)

[Calculate the Number of Leaf Pages in the Index \[page 92\]](#)

[Calculate the Number of Non-Leaf Pages in the Index \[page 92\]](#)

[Calculate the Total Number of Non-Leaf Index Pages \[page 93\]](#)

[Calculate Allocation Overhead and Total Pages \[page 93\]](#)

4.6.5.1 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.

4.6.5.1.1 Fixed-Length Columns Only

A formula to use if the table contains only fixed-length columns defined as NOT NULL.

The formula is:

$$\begin{array}{r}
 6 \text{ (Overhead)} \\
 + \quad \text{Sum of bytes in all fixed-length columns} \\
 \hline
 \text{Data row size}
 \end{array}$$

i Note

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.

4.6.5.1.2 Some Variable-Length Columns

A formula to use if the table contains variable-length columns or columns that allow NULL values.

The formula is:

```
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
```

For example:

```
8
+ 100
+ 50
+ 4
-----
162
```

4.6.5.2 Compute the Number of Data Pages

Formulas to calculate the number of data pages.

The formulas are:

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

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

For example:

```
2002 / 162 = 12 data rows per page
```

```
9,000,000 / 12 = 750,000 data pages
```

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

4.6.5.3 Calculate Allocation Overhead and Total Pages

Formula to calculate the 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

For example:

750,000 / 63,750 = 12

750,000 / 2000 = 375

4.6.5.3.1 Total Pages Needed

A formula to add the number of OAM pages to earlier totals to determine the total number of pages required.

The formula is:

	Minimum	Maximum
Data pages	+	+
OAM pages	+	+
<hr/>		
Total		

For example:

Minimum	Maximum
+750000	750000
+ 12	375
<hr/>	
750012	750375

4.6.5.4 Calculate the Size of the Index Row

Formulas for clustered and nonclustered indexes on data-only-length tables. Index rows containing variable-length columns require more overhead than index rows containing only fixed-length values.

4.6.5.4.1 Fixed-Length Keys Only

A formula to use if the index contains only fixed-length keys defined as NOT NULL.

The formula is:

9 (Overhead)
+ Sum of fixed-length keys
<hr/>
Size of index row

4.6.5.4.2 Some Variable-Length Keys

A formula to use if the index contains any variable-length keys or columns that allow NULL values.

The formula is:

$$\begin{array}{r} 9 \text{ (Overhead)} \\ + \text{ Sum of length of fixed-length keys} \\ + \text{ Sum of length of variable-length keys} \\ + \text{ Number of variable-length keys} * 2 \\ \hline \text{Size of index rows} \end{array}$$

For example:

$$\begin{array}{r} 9 \\ + 4 \\ + 20 \\ + 2 \\ \hline 35 \end{array}$$

4.6.5.5 Calculate the Number of Leaf Pages in the Index

Formulas to use to calculate the number of leaf pages in the index.

The formulas are:

$$2002 / \text{Size of index row} = \text{No. of rows per page}$$

$$\text{No. of rows in table} / \text{No. of rows per page} = \text{No. of leaf pages}$$

For example:

$$2002 / 35 = 57 \text{ Nonclustered index rows per page}$$

$$9,000,000 / 57 = 157,895 \text{ leaf pages}$$

4.6.5.6 Calculate the Number of Non-Leaf Pages in the Index

A formula to calculate the number of non-leaf pages in the index.

The formula is:

$$\text{No. of leaf pages} / \text{No. of index rows per page} = \text{No. of pages at next level}$$

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 index pages at previous level / No. of non-leaf index rows per page = No.
of index pages at next level
```

For example:

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

4.6.5.7 Calculate the Total Number of Non-Leaf Index Pages

A formula to use to add the number of pages at each level to determine the total number of pages in the index.

The formula is:

```
Index levels    Pages
3
2              +
1              +
0              +
-----
Total number of 2K pages used
```

For example:

```
Pages    Rows
1        49
49       2771
2771     157895
157895   9000000
-----
160716
```

4.6.5.8 Calculate Allocation Overhead and Total Pages

Formulas to calculate allocation overhead and total pages.

The formulas are:

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

```
Number of index pages / 2000 = Maximum OAM pages
```

For example:

```
160713 / 63,750 = 3 (minimum)
```

160713 / 2000 = 81 (maximum)

4.6.5.8.1 Total Pages Needed

A formula to add the number of OAM pages to the total determine the total number of index pages.

The formula is:

	Minimum	Maximum
Nonclustered index pages	+	+
OAM pages	+	+
<hr/>		
Total		

For example:

	Minimum	Maximum
+	160716	160716
+	3	81
<hr/>		
	160719	160797

4.6.6 Other Factors Affecting Object Size

In addition to the effects of data modifications that occur over time, other factors can affect object size and size estimates.

Such as:

- Space management properties
- Whether computations used average row size or maximum row size
- Very small text rows
- Use of `text` and image data

4.6.6.1 Effects of Space Management Properties

Values for `fillfactor`, `exp_row_size`, `reservepagegap` and `max_rows_per_page` can affect object size.

4.6.6.1.1 fillfactor

The `fillfactor` you specify for `create index` is applied when 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 with `alter table` 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.

Related Information

[Index Maintenance Reduction \[page 50\]](#)

4.6.6.1.2 `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 increases 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.

Related Information

[Row Forwarding Reduction \[page 56\]](#)

4.6.6.1.3 `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.

Related Information

[Space for Forwarded Rows and Inserts \[page 61\]](#)

4.6.6.1.4 max_rows_per_page

The `max_rows_per_page` value (specified by `create index`, `create table`, `alter table`, or `sp_chgattribute`) limits the number of rows on a data page.

To compute the correct values when using `max_rows_per_page`, use the `max_rows_per_page` value or the computed number of data rows per page, whichever is smaller, when you compute the number of data pages and calculate the number of leaf pages in the index..

Related Information

[Compute the Number of Data Pages \[page 80\]](#)

[Calculate the Number of Leaf Pages in the Index \[page 85\]](#)

[max_rows_per_page Usage on allpages-locked Tables \[page 68\]](#)

4.6.7 Very Small Rows

For all-pages locked tables, SAP ASE cannot store more than 256 data or index rows on a page.

Even if your rows are extremely short, the minimum number of data pages is:

Number of Rows / 256 =	Number of data pages required
------------------------	-------------------------------

4.6.8 LOB Pages

Each `text` or `image` or Java off-row column stores a 16-byte pointer in the data row with the datatype `varbinary(16)`.

The number of bytes each LOB page stores depends on the server's logical page size:

- 2k – 1800 bytes
- 4k – 3600 bytes
- 8k – 7650 bytes
- 16k – 16200 bytes

Each column that is initialized requires at least 2K (one data page on a 2K server) of storage space

Columns store implicit NULL values, meaning that the text pointer in the data row remains NULL and no text page is initialized for the value, saving at least 2K of storage space.

If a LOB column is defined to allow NULL values, and the row is created with an `insert` statement that includes NULL for the column, the column is not initialized, and the storage is not allocated.

If a LOB column is changed in any way with `update`, then the text page is allocated. Inserts or updates that place actual data in a column initialize the page. If the column is subsequently set to NULL, a single page remains allocated.

Each LOB page stores approximately 1800 to 16200 bytes of data, depending on the server's logical page size. To estimate the number of pages that a particular entry will use, use this formula (alter the formula according to your server's logical page size):

$$\text{Data length} / 1800 = \text{Number of 2K pages}$$

The result should be rounded up in all cases; that is, a data length of 1801 bytes requires two 2K pages.

The total space required for the data may be slightly larger than the calculated value, because some LOB pages store pointer information for other page chains in the column. SAP ASE uses this pointer information to perform random access and prefetch data when accessing LOB columns. The additional space required to store pointer information depends on the total size and type of the data stored in the column.

Data Size and Type	Additional Pages Required for Pointer Information
400K image	0 to 1 page
700K image	0 to 2 pages
5MB image	1 to 11 pages
400K of multibyte text	1 to 2 pages
700K of multibyte text	1 to 3 pages
5MB of multibyte text	2 to 22 pages

4.6.9 Advantages of Using Formulas to Estimate Object Size

There are several advantages of using formulas to estimate the object size.

The advantages are:

- You learn more details of the internals of data and index storage.
- The formulas provide flexibility for specifying averages sizes for character or binary columns.
- While computing the index size, you see how many levels each index has, which helps estimate performance.

4.6.10 Disadvantages of Using Formulas to Estimate Object Size

There are a few disadvantages of using formulas to estimate the object size.

The disadvantages are:

- The estimates are only as good as your estimates of average size for variable-length columns.
- The multistep calculations are complex, and skipping steps may lead to errors.
- The actual size of an object may be different from the calculations, based on use.

5 Database Maintenance

Learn how to improve the performance of maintenance tasks since maintenance activities can affect the performance of other SAP ASE activities.

Maintenance activities include tasks such as dropping and recreating indexes, performing `dbcc` checks, and updating table and index statistics. All of these activities can compete with other processing work on the server.

Whenever possible, perform maintenance tasks when your SAP ASE usage is low. This chapter can help you determine the impact these activities have on individual application performance, and on overall SAP ASE performance.

5.1 Run reorg on Tables and Indexes

The `reorg` command can improve performance for data-only-locked tables by improving the space utilization for tables and indexes.

The `reorg` subcommands and their uses are:

- `reclaim_space` – clears committed deletes and the space that is left when updates shorten the length of data rows.
- `forwarded_rows` – returns forwarded rows to home pages.
- `compact` – performs both of the operations above.
- `rebuild` – rebuilds an entire table or index. You can use `reorg rebuild` on both all-pages and data-only locked tables.
- `reorg defrag` – allows you to schedule and resume reorganization, also allowing concurrent reads or writes on the data being reorganized.

When you run `reorg rebuild` on a table, and the table is locked for the entire time it takes to rebuild the table and its indexes. Schedule the `reorg rebuild` command on a table when users do not need access to the table.

All of the other `reorg` commands, including `reorg rebuild` on an index, lock a small number of pages at a time, and use short, independent transactions to perform their work. You can run these commands at any time. The only negative effect might be on systems that are very I/O bound.

5.2 Create and Maintain Indexes

When a user creates an index, all other users are locked out of the table. The type of lock depends on the type of index.

- Creating a clustered index requires an exclusive table lock, locking out all table activity. Since rows in a clustered index are arranged in order by the index key, `create clustered index` reorders data pages.
- Creating a nonclustered index requires a shared table lock, locking out update activity.

5.2.1 Configure SAP ASE to Speed Sorting

Use the `number of sort buffers` configuration parameter to set the number of buffers that can be used in cache to hold pages from the input tables. In addition, parallel sorting can benefit from large I/O in the cache used to perform the sort.

See Chapter 5, “Parallel Query Processing” in *Performance and Tuning Series: Query Processing and Abstract Plans*.

5.2.2 Dump the Database After Creating an Index

To recover a database that you have not dumped since you created the index, the entire `create index` process is executed again while loading transaction log dumps.

When you create an index, SAP ASE writes the `create index` transaction and the page allocations to the transaction log, but does not log the actual changes to the data and index pages.

If you routinely re-create indexes (for example, to maintain the `fillfactor` in the index), you may want to schedule these operations to run shortly before a routine database dump.

5.2.3 Create an Index on Sorted Data

To recreate a clustered index, or to create one on data that was bulk copied into the server in index key order, use the `sorted_data` option to `create index` to shorten index creation time.

Since the data rows must be arranged in key order for clustered indexes, creating a clustered index without `sorted_data` requires you to rewrite the data rows to a complete new set of data pages. In some cases, SAP ASE can skip sorting and copying the table's data rows: Factors include table partitioning and `on` clauses used in the `create index` statement.

When you are creating an index on a nonpartitioned table, `sorted_data` and the use of any of the following clauses requires you to copy the data, but does not require a sort:

- `ignore_dup_row`

- `fillfactor`
- The `on <segment_name>` clause, specifying a different segment from the segment where the table data is located
- The `max_rows_per_page` clause, specifying a value that is different from the value associated with the table

When these options and `sorted_data` are included in a `create index` on a partitioned table, the sort step is performed and the data is copied, distributing the data pages evenly on the table's partitions.

Table 4: Using Options for Creating a Clustered index

Options	Partitioned Table	Unpartitioned Table
No options specified	Parallel sort; copies data, distributing evenly on partitions; creates index tree.	Either parallel or nonparallel sort; copies data, creates index tree.
with <code>sorted_data</code> only or with <code>sorted_data on <same_segment></code>	Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.	Creates index tree only. Does not perform the sort or copy data. Does not run in parallel.
with <code>sorted_data</code> and <code>ignore_dup_row</code> or <code>fillfactor</code> or on <code><other_segment ></code> or <code>max_rows_per_page</code>	Parallel sort; copies data, distributing evenly on partitions; creates index tree.	Copies data and creates the index tree. Does not perform the sort. Does not run in parallel.

In the simplest case, using `sorted_data` and no other options on a nonpartitioned table, the order of the table rows is checked and the index tree is built during this single scan.

If the data rows must be copied, but no sort needs to be performed, a single table scan checks the order of rows, builds the index tree, and copies the data pages to the new location in a single table scan.

For large tables that require numerous passes to build the index, saving the sort time considerably reduces I/O and CPU utilization.

When you create a clustered index that copies the data rows, the space available must be approximately 120 percent of the table size to copy the data and store the index pages.

5.2.4 Maintain Index and Column Statistics

The histogram and density values for an index are not maintained as data rows are added and deleted. The database owner must issue an `update statistics` command to ensure that statistics are current.

Run `update statistics` after:

- Deleting or inserting rows that change the skew of key values in the index.
- Adding rows to a table for which rows were previously deleted with `truncate table`.
- Updating values in index columns.
- Inserts to any index that includes an IDENTITY column or any increasing key value. Date columns often have regularly increasing keys.

Running `update statistics` on these types of indexes is especially important if the `IDENTITY` column or other increasing key is the leading column in the index. After a number of rows have been inserted past the last key in the table when the index was created, all that the optimizer can tell is that the search value lies beyond the last row in the distribution page. It cannot accurately determine how many rows match a given value.

i Note

Failure to update statistics can severely impair performance.

See *Performance and Tuning Series: Improving Performance with Statistical Analysis*.

5.2.5 Rebuild Indexes

Rebuilding indexes reclaims space in the binary trees (a tree where all leaf pages are the same distance from the root page of the index). As pages are split and rows are deleted, indexes may contain many pages that contain only a few rows.

Also, if the application performs scans on covering nonclustered indexes and large I/O, rebuilding the nonclustered index maintains the effectiveness of large I/O by reducing fragmentation.

You can rebuild indexes by dropping and recreating the index.

Rebuild indexes when:

- Data and usage patterns have changed significantly.
- A period of heavy inserts is expected, or has just been completed.
- The sort order has changed.
- Queries that use large I/O require more disk reads than expected, or `optdiag` reports lower cluster ratios than usual.
- Space usage exceeds estimates because heavy data modification has left many data and index pages partially full.
- Space for expansion provided by the space management properties (`fillfactor`, expected row size, and reserve page gap) has been filled by inserts and updates, resulting in page splits, forwarded rows, and fragmentation.
- `dbcc` has identified errors in the index.

If you recreate a clustered index or run `reorg rebuild` on a data-only-locked or all-pages-locked table, all nonclustered indexes are recreated, since creating the clustered index moves rows to different pages.

When system activity is low:

- Delete all indexes to allow more efficient bulk inserts.
- Create a new group of indexes to help generate a set of reports.

5.3 Create or Alter a Database

Creating or altering a database is I/O-intensive; consequently, other I/O-intensive operations may suffer. When you create a database, SAP ASE copies the `model` database to the new database and then initializes all the allocation pages and clears database pages.

To speed database creation or minimize its impact on other processes:

- Use the `create database...for load` option if you are restoring a database; that is, if you are getting ready to issue a `load database` command.
When you create a database without `for load`, SAP ASE copies `model` and then initializes all of the allocation units.
When you use `for load`, SAP ASE does initialize the allocation units until the load is complete. Then it initializes only the untouched allocation units. If you are loading a very large database dump, this can save a lot of time.
- Create databases during off-peak hours if possible.

`create database` and `alter database` perform concurrent, parallel I/O when clearing database pages. The number of devices is limited by the `number of large i/o buffers` configuration parameter. The default value for this parameter is 6, allowing parallel I/O on 6 devices at once.

A single `create database` and `alter database` command can use up to 32 of these buffers at once. These buffers are also used by `load database`, disk mirroring, and some `dbcc` commands.

Using the default value of 6, if you specify more than 6 devices, the first 6 writes are immediately started. As the I/O to each device completes, the 16K buffers are used for remaining devices listed in the command. The following example names 10 separate devices:

```
create database hugeadb
  on dev1 = 100,
  dev2 = 100,
  dev3 = 100,
  dev4 = 100,
  dev5 = 100,
  dev6 = 100,
  dev7 = 100,
  dev8 = 100
log on logdev1 = 100,
  logdev2 = 100
```

During operations that use these buffers, a message is sent to the log when the number of buffers is exceeded. This information, for the `create database` command above, shows that `create database` started clearing devices on the first 6 disks, using all of the large I/O buffers, and then waited for them to complete before clearing the pages on other devices:

```
CREATE DATABASE: allocating 51200 pages on disk 'dev1'
CREATE DATABASE: allocating 51200 pages on disk 'dev2'
CREATE DATABASE: allocating 51200 pages on disk 'dev3'
CREATE DATABASE: allocating 51200 pages on disk 'dev4'
CREATE DATABASE: allocating 51200 pages on disk 'dev5'
CREATE DATABASE: allocating 51200 pages on disk 'dev6'
01:00000:00013:1999/07/26 15:36:17.54 server No disk i/o buffers are available
for this operation. The total number of buffers is controlled by the
configuration parameter 'number of large i/o buffers'.
CREATE DATABASE: allocating 51200 pages on disk 'dev7'
CREATE DATABASE: allocating 51200 pages on disk 'dev8'
CREATE DATABASE: allocating 51200 pages on disk 'logdev1'
```

```
CREATE DATABASE: allocating 51200 pages on disk 'logdev2'
```

i Note

In SAP ASE version 12.5.0.3 and later, the size of the large I/O buffers used by `create database`, `alter database`, `load database`, and `dbcc checkalloc` is one allocation (256 pages), not one extent (8 pages), as it was in earlier versions. The server thus requires more memory allocation for large buffers. For example, a disk buffer that required memory for 8 pages in earlier versions now requires memory for 256 pages.

5.4 Backup and Recovery

All SAP ASE backups are performed by Backup Server. The backup architecture uses a client/server paradigm, with SAP ASE servers as clients to Backup Server.

5.4.1 Local Backups

SAP ASE sends the local Backup Server instructions, via remote procedure calls, telling the Backup Server which pages to dump or load, which backup devices to use, and other options. Backup Server performs all the disk I/O.

SAP ASE does not read or send dump and load data, it sends only instructions.

5.4.2 Remote Backups

Backup Server also supports backups to remote machines. For remote dumps and loads, a local Backup Server performs the disk I/O related to the database device and sends the data over the network to the remote Backup Server, which stores it on the dump device.

5.4.3 Online Backups

You can perform backups while a database is active. Clearly, such processing affects other transactions, but you should not hesitate to back up critical databases as often as necessary to satisfy the reliability requirements of the system.

See the *System Administration Guide, Volume 2* for a complete discussion of backup and recovery strategies.

5.4.4 Use Thresholds to Prevent Running Out of Log Space

If your database has limited log space, and you occasionally hit the *last-chance threshold*, install a second threshold that provides ample time to perform a transaction log dump.

Running out of log space has severe performance impacts. Users cannot execute any data modification commands until log space has been freed.

5.4.5 Minimize Recovery Time

You can help minimize recovery time by changing the `recovery interval` configuration parameter.

The default value of 5 minutes per database works for most installations. Reduce this value only if functional requirements dictate a faster recovery period. Reducing the value increases the amount of I/O required.

See Chapter 5, “Memory Use and Performance,” in *Performance and Tuning Series: Basics*.

Recovery speed may also be affected by the value of the `housekeeper free write percent` configuration parameter. The default value of this parameter allows the server’s housekeeper wash task to write dirty buffers to disk during the server’s idle cycles, as long as disk I/O is not increased by more than 20 percent.

5.4.6 Recovery Order

During recovery, system databases are recovered first. Then, user databases are recovered in order by database ID.

5.5 Bulk-Copy

Bulk-copying into a table on SAP ASE runs fastest when there are no clustered indexes on the table and you have enabled `select into/ bulkcopy`.

If you have not enabled this option, `slow bcp` is used for tables with any index or active trigger.

`fast bcp` logs page allocation only for tables without an index. `fast bcp` saves time because it does not update indexes for each data insert, nor does it log the changes to the index pages. However, if you use `fast bcp` on a table with an index, it does log index updates.

`fast bcp` is automatically used for tables with triggers. To use `slow bcp`, disable the `select into/bulk copy` database option while you perform the copy.

To use fast bulk-copy:

1. Use `sp_dboption` to set the `select into/bulkcopy/pllsort` option. Remember to disable the option after the bulk-copy operation completes.

2. Drop any clustered indexes. Recreate them when the bulk-copy completes.

i Note

You need not deactivate triggers during the copy.

During fast bulk-copy, rules are not enforced, but defaults are.

Since changes to the data are not logged, perform a `dump database` soon after a fast bulk-copy operation. Performing a fast bulk-copy in a database blocks the use of `dump transaction`, since the unlogged data changes cannot be recovered from the transaction log dump.

5.5.1 Parallel Bulk-Copy

For fastest performance, use fast bulk-copy to copy data into partitioned tables. For each bulk-copy session, specify the partition on which the data should reside.

If your input file is already in sorted order, you can bulk-copy data into partitions in order, and avoid the sorting step while creating clustered indexes.

See Chapter 10, “Partitioning Tables and Indexes,” in the *Transact-SQL Users Guide* for step-by-step procedures.

5.5.2 Batches and Bulk-Copy

If you specify a batch size during a fast bulk-copy, each new batch must start on a new data page, since only the page allocations, and not the data changes, are logged during a fast bulk-copy.

Copying 1000 rows with a batch size of 1 requires 1000 data pages and 1000 allocation records in the transaction log.

If you use a small batch size to help detect errors in the input file, you may want to choose a batch size that corresponds to the numbers of rows that fit on a data page.

5.5.3 Slow Bulk-Copy

By default, SAP ASE uses `slow bcp` by default if a table has a clustered index, index, or trigger with the `select into/bulk copy enabled`.

For slow bulk-copy:

- You do not have to set `select into/bulkcopy`.
- Rules are not enforced and triggers are not fired, but defaults are enforced.
- All data changes are logged, as are page allocations.
- Indexes are updated as rows are copied in, and index changes are logged.

5.5.4 Improve Bulk-Copy Performance

Additional ways to increase bulk-copy performance.

- Set the `trunc log on chkpt` option to keep the transaction log from filling up. If your database has a threshold procedure that automatically dumps the log when it fills, you save the transaction dump time. Each batch is a separate transaction, so if you do not specify a batch size, setting `trunc log on chkpt` does not improve performance.
- Set the `number of pre-allocated extents` configuration parameter high if you perform many large bulk copies.
See *Reference Manual: Configuration Parameters*.
- Find the optimal network packet size.
See *Performance and Tuning Series: Basics > Networks and Performance*.

5.5.5 Replacing the Data in a Large Table

If you are replacing all the data in a large table, use `truncate table`, which performs reduced logging, instead of `delete`. Only page deallocations are logged.

Procedure

1. Truncate the table.
2. Drop all indexes on the table.
3. Load the data.
4. Recreate the indexes.

Results

See the *Reference Manual: Commands*.

5.5.6 Add Large Amounts of Data to a Table

When you are adding 10 – 20 percent or more to a large table, drop the nonclustered indexes, load the data, and then recreate nonclustered indexes.

For very large tables, you may need to leave the clustered index in place due to space constraints. SAP ASE must make a copy of the table when it creates a clustered index. In many cases, once tables become very large, the time required to perform a slow bulk-copy with the index in place is less than the amount of time it takes to perform a fast bulk-copy and recreate the clustered index.

5.5.7 Use Partitions and Multiple Bulk-Copy Processes

If you load data into a table without indexes, you can create partitions on the table and use one `bcp` session for each partition.

See Chapter 4, “Using `bcp` to Transfer Data to and from SAP ASE” in the *Utility Guide*.

5.5.8 Impact on Other Users

Bulk-copying large tables in or out may affect response time for other users.

If possible:

- Schedule bulk-copy operations for off-peak hours.
- Use fast bulk-copy, since it performs less logging and less I/O.

5.6 Database Consistency Checker

Periodically, use `dbcc` to run database consistency checks. If you back up a corrupt database, the backup is useless. `dbcc` affects performance, since `dbcc` must acquire locks on the objects it checks.

See Chapter 10, “Checking Database Consistency” in the *System Administration Guide: Volume 2* for information about `dbcc` and locking, with additional information about how to minimize the effects of `dbcc` on user applications.

5.7 Use `dbcc tune (cleanup)`

SAP ASE performs redundant memory cleanup checking as a final integrity check after processing each task. In very high throughput environments, you may realize a slight performance improvement by skipping this cleanup error check.

To turn off error checking, enter:

```
dbcc tune (cleanup, 1)
```

The final cleanup frees any memory a task might hold. If you turn error checking off, but you get memory errors, reenable the checking by entering:

```
dbcc tune (cleanup, 0)
```

5.8 Use dbcc tune on Spinlocks

When a scaling problem results from spinlock contention, use `des_bind` to improve the scalability when object descriptors are reserved for hot objects.

Descriptors for bound objects are never released. Binding the descriptors for even a few commonly used objects may reduce the overall metadata spinlock contention and improve performance.

```
dbcc tune(des_bind, <dbid>, <objname>)
```

To remove the binding, use:

```
dbcc tune(des_unbind, <dbid>, <objname>)
```

i Note

There cannot be users in the database when you run `dbcc tune des_unbind` to unbind an object from a database.

i Note

`dbcc tune(hotdes, <dbid>)` prints the DES structures of bound objects, with the object names listed under the `objname= label`.

Do not use `des_bind`:

- On objects in system databases such as `master` and `tempdb`
- On system tables

Generally, you cannot run operations that exclusively use an object with a bound DES. For example, you cannot:

- Drop bound objects
- Run these commands on bound tables:
 - `reorg rebuild`
 - `create index`
 - `create trigger`
 - `create constraint`
 - `drop index`
 - `drop trigger`
 - `drop constraint`
- Run most `alter table` operations

Databases containing an object with a bound DES:

- Cannot be put in single-user mode
- Cannot be dropped

Since `des_bind` is not persistent, you must reissue any binding commands each time you restart the server.

5.9 Determine the Space Available for Maintenance Activities

Several maintenance operations require room to make a copy of the data pages of a table.

These are:

- `create clustered index`
- `alter table...lock`
- Some `alter table` commands that add or modify columns
- `alter table...partition by`
- `reorg rebuild` on a table

In most cases, these commands also require space to recreate any indexes, so you must determine:

- 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

5.9.1 Overview of Space Requirements

Any command that copies a table's rows also recreates all of the indexes on the table. You need enough available space for a complete copy of the table and copies of all indexes.

These commands do not estimate how much space is needed. If a command runs out of space on any segment used by the table or its indexes the command stops, and issues an error message. For large tables, can occur minutes, 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 alter 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.

5.9.2 Check Space Usage and Available Space

As a simple guideline, copying a table and its indexes requires space equal to the current space used by the table and its indexes, plus about 20% additional space.

However:

- If data modifications have created many partially full pages, the space requirement for the copy of the table can be smaller than the current size.
- If space-management properties for the table have changed, or if space required by `fillfactor` or `reservepagegap` has been filled by data modifications, the size required for the copy of the table can be larger.
- Adding columns or modifying columns to larger datatypes requires more space for the copy.

Log space is also required. Because SAP ASE processes `reorg rebuild` as a single transaction, the amount of log space required can be large, particularly if the table it is rebuilding has multiple nonclustered indexes. Each nonclustered index requires log space, and there must be sufficient log space to create all indexes.

5.9.2.1 Check Space Used for Tables and Indexes

To see the size of a table and its indexes, use `sp_spaceused`.

The syntax is:

```
sp_spaceused titles, 1
```

Related Information

[Calculating the Sizes of Data-Only-Locked Tables \[page 88\]](#)

5.9.2.2 Check Space on Segments

Tables are always copied to free space on the segment where they are currently stored, and indexes are recreated on the segment where they are currently stored. Commands that create clustered indexes can specify a segment.

The copy of the table and the clustered index are created on the target segment.

To determine the number of pages available on a segment, use `sp_helpsegment`. The last line of `sp_helpsegment` shows the total number of free pages available on a segment.

This command prints segment information for the `default` segment, where objects are stored when no segment was explicitly specified:

```
sp_helpsegment "default"
```

`sp_helpsegment` reports the names of indexes on the segment. 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 by `sp_help`.

5.9.2.3 Check 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
```

5.9.2.4 Space Management Properties Applied to the Table

Space management properties for the table during the copy process.

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.

5.9.2.5 Space Management Properties Applied to the Index

When indexes are rebuilt, space management properties for the indexes are applied.

They are applied as follows:

- If `sp_chgattribute` has been used to save `fillfactor` values for indexes, these values are applied when the indexes are recreated.
- If `reservepagegap` values are set for indexes, these values are applied when the indexes are recreated.

5.9.3 Estimate the Effects of Space Management Properties

A table showing how to estimate the effects of setting space management properties.

Property	Formula	Example
<code>fillfactor</code>	Requires $(100/\text{fillfactor}) * \langle \text{num_pages} \rangle$ if pages are currently fully packed	<code>fillfactor</code> of 75 requires 1.33 times current number of pages; a table of 1,000 pages grows to 1,333 pages.
<code>reservepagegap</code>	Increases space by $1/\text{reservepagegap}$ if extents are currently filled	<code>reservepagegap</code> of 10 increase space used by 10%; a table of 1,000 pages grows to 1,100 pages.
<code>max_rows_per_page</code>	Converted to <code>exp_row_size</code> when converting to data-only-locking	See the following table.
<code>exp_row_size</code>	Increase depends on number of rows smaller than <code>exp_row_size</code> , and the average length of those rows	If <code>exp_row_size</code> 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.

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.

`exp_row_size` is enforced during the copy. This table shows how the values are converted.

If <code>max_rows_per_page</code> is Set to	Set <code>exp_row_size</code> to
0	Percentage value set by default <code>exp_row_size</code> percent
1 – 254	The smaller of: <ul style="list-style-type: none"> • Maximum row size • 2K logical page – $2002/\text{max_rows_per_page}$ value 4K logical page – $4050/\text{max_rows_per_page}$ value 8K logical page – $8146/\text{max_rows_per_page}$ value 16K logical page – $16338/\text{max_rows_per_page}$ value

Related Information

[Setting Space Management Properties \[page 50\]](#)

5.9.4 If There is Not Enough Space

If you do not have enough space to copy the table and recreate all the indexes, determine whether dropping the nonclustered indexes on the table leaves enough room to create a copy of the table.

Without any nonclustered indexes, the copy operation requires space just for the table and the clustered index.

Do not drop the clustered index, since it is used to order the copied rows, and attempting to recreate it later may require space to make a copy of the table. Recreate the nonclustered indexes when the command completes.

6 Temporary Databases

This chapter discusses performance issues associated with temporary databases. Temporary databases are server-wide resources, and are used primarily for processing sorts, creating worktables, reformatting, and storing temporary tables and indexes created by users. Anyone can create objects in temporary databases. Many processes use them silently.

Many applications use stored procedures that create tables in temporary databases to expedite complex joins or to perform other complex data analysis that cannot be performed easily in a single step.

6.1 How Temporary Database Management Affects Performance

Good management of temporary databases is critical to the overall performance of SAP ASE. However, temporary tables can add to the size requirement of `tempdb`.

Using temporary tables greatly affects the performance of SAP ASE and your applications. You cannot overlook the management of temporary databases or leave them in a default state. On many servers, `tempdb` is the most dynamic database.

You can avoid most of the performance issues with temporary databases by planning in advance, and taking these issues into consideration:

- Temporary databases fill frequently, generating error messages to users, who must then resubmit their queries when space becomes available.
- Temporary databases sort slowly and queries against them display uneven performance.
- User queries are often temporarily blocked from creating temporary tables because of locks on system tables.
- Heavily used objects in a temporary database flush other pages out of the data cache.

Resolve these issues by:

- Configuring a sufficient number of user temporary databases.
- Sizing temporary databases correctly for all SAP ASE activity
- Placing temporary databases optimally to minimize contention
- Minimizing the resource locking within temporary databases
- Binding temporary databases to their own data cache
- Configuring temporary database groups correctly
- Binding logins and applications to the appropriate temporary database or group.

6.2 Use Temporary Tables

Tables created in temporary database are called temporary tables. Use the temporary database to create different types of temporary tables. The types of temporary tables are:

- Hashed (#) temporary tables
- Regular user tables
- Worktables

6.2.1 Hashed (#) Temporary Tables

Hashed temporary tables exist only for the duration of the user session or for the scope of the procedure that creates them, and can be either manually or automatically dropped at the end of the session or procedure.

Hashed temporary tables:

- Cannot be shared between user connections
- Are created in the temporary database assigned for the session.

Create hashed temporary tables by including a hash mark (“#”) as the first character of the table name:

```
create table #temptable (...)
```

or:

```
select <select_list>  
into #temptable ...
```

When you create indexes on temporary tables, the indexes are stored in the same session assigned to the temporary database where the hashed table resides:

```
create index littletableix on #littletable(coll)
```

6.2.2 Regular User Tables

To create regular user tables in a temporary table, specify the database name in the `create table` command.

The syntax is:

```
create table tempdb..temptable (...)
```

Regular user tables in the temporary database:

- Can persist across sessions
- Can be used by bulk copy (`bcp`) operations
- Can be shared by granting permissions on them
- Must either be explicitly dropped by the owner or are automatically removed when SAP ASE is restarted

or:

```
select <select_list>  
into tempdb..temptable
```

You can create indexes on regular user tables created in the temporary database:

```
create index tempix on tempdb..temptable(coll)
```

6.2.3 Worktables

SAP ASE creates internal temporary tables for the session-assigned `tempdb` for merges, sorts, joins, and so on. These temporary tables are called worktables.

Worktables:

- Are never shared
- Disappear as soon as the command completes

6.3 Temporary Databases

To avoid performance concerns that result from using a single temporary databases, you can create multiple temporary databases.

SAP ASE includes one system-created temporary database called `tempdb`, which is created on the master device when you install SAP ASE.

In addition to `tempdb`, SAP ASE allows users to create multiple temporary databases. User-created temporary databases are similar to the system `tempdb`: they are used primarily to create temporary objects, and are recreated instead of recovered during start-up. Unlike `tempdb`, you can drop user-created temporary databases.

Multiple temporary databases:

- Reduce contention on system catalogs and log files in the system `tempdb`
- Can be created on fast access devices
- Can be created or dropped as needed.

6.4 Session-Assigned Temporary Database

When a client connects, SAP ASE assigns a temporary database to its session.

SAP ASE uses this session-assigned temporary database as a default space where it creates temporary objects (including hashed-temporary tables and worktables) for work the client performs. The session-assigned temporary database remains assigned to the session until the session connects to the client.

SAP ASE selects temporary databases for a session according to these rules:

- If a binding already exists for a login, that binding is used.
- If an application name is specified and it has a binding, use that binding.
- If SAP ASE does not find a binding, it assigns a temporary database from the default group using a round-robin scheme.

To specify that SAP ASE creates an object in a specific temporary database. For example:

```
create procedure inv_amounts as
  select stor_id, "Total Due" = sum(amount)
  from #tempstores
  group by stor_id
```

6.5 Create User Temporary Databases

Use the `temporary database` keyword in the `create database` syntax to create multiple temporary databases.

The syntax is:

```
create temporary database <temporary_database_name> on <device_name>=size log on
<device_name>=size
```

For example, to create a user temporary database named `tempdb_1` on the `tempdb_device`, enter:

```
create temporary database tempdb_1 on tempdb_device = 3
log on log_device = 1
```

6.6 Configure a Default tempdb Group

SAP ASE includes a group of temporary databases called the default group.

When SAP ASE starts a session, it selects a temporary database from the default group (using a round-robin technique) in which all temporary database activities are performed. SAP ASE assigns this temporary database to the session. `sp_who` displays this temporary database in the `tempdbname` column. The round-robin scheme allows SAP ASE to distribute the load evenly across all temporary databases in the default group because a single temporary database from the group is not performing all activities.

Initially, the default group consists only of `tempdb`. However, users may add multiple user databases to the default group. Use `sp_tempdb` to add a user database to the default group. For example, to add `tempdb_1` to the default group, use:

```
sp_tempdb "add", "tempdb_1" , "default"
```

To drop `tempdb_1` from the default group, use:

```
sp_tempdb "drop", "tempdb_1" , "default"
```

See the *Reference Manual: Procedures* for the complete `sp_tempdb` syntax.

6.7 Bind to Groups and tempdb

The `sp_tempdb . . . 'bind' . . . 'unbind'` system procedure allows you to bind, or unbind, an application or login to specific temporary database or tempdb group.

After you create the binding, when the application or login connects to the server, SAP ASE assigns the specified temporary database or temporary database group to which it is bound. Binding allows you to control the temporary database assignments for specific applications or logins.

This example binds the log in sa to the default group:

```
sp_tempdb 'bind', 'lg', 'sa', 'GR', 'default'
```

This example unbinds the login sa:

```
sp_tempdb 'unbind', 'lg', 'sa'
```

See Reference Manual: Procedures for the complete `sp_tempdb` syntax.

6.7.1 Bind applications and Logins to Temporary Databases

Identify your application and login requirements for temporary databases. Bind these applications and logins to different databases or default groups to distribute the load evenly across available temporary databases to avoid catalog contention.

Inappropriate bindings do not solve catalog contention even if there is a sufficient number of temporary databases—SAP ASE may not distribute the load evenly across the temporary databases.

Related Information

[Bind to Groups and tempdb \[page 119\]](#)

6.8 Tune System Temporary Databases for Performance

Solutions to configuration issues related to temporary databases.

6.8.1 Placing System tempdb

Consideration to take into account when placing the system `tempdb`.

When deciding where to place `tempdb`:

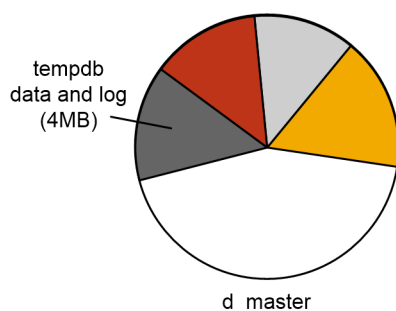
- Keep `tempdb` on separate physical disks than your critical application databases.
- Use the fastest disks available. If your platform supports solid state devices and `tempdb` use is a bottleneck for your applications, use those devices.
- After you expand `tempdb` onto additional devices, drop the master device from the `system`, `default`, and `logsegment` segments.

Although you can expand `tempdb` on the same device as the `master` database, SAP suggests that you use separate devices. Also, remember that logical devices, but not databases, are mirrored using SAP ASE mirroring. If you mirror the master device, you create a mirror of all portions of the databases that reside on the master device. If the mirror uses `serial` writes, this can have a serious performance impact if `tempdb` is heavily used.

6.8.1.1 Initial Allocation of System tempdb

When you install SAP ASE, the size of `tempdb` depends on the server's to logical page size, and is located completely on the master device.

`tempdb` is typically the first database that a system administrator needs to make larger. The more users on the server, the larger it needs to be. Depending on your needs, you may want to stripe `tempdb` across several devices.



Use `sp_helpdb` to see the size and status of `tempdb`. The following example shows `tempdb` defaults at installation time:

```
sp_helpdb tempdb
```

name	db_size	owner	dbid	created	status
tempdb	2.0 MB	sa		2 May 22, 1999	select into/bulkcopy

device_frag	size	usage	free kbytes
master	2.0 MB	data and log	1248

6.8.1.2 Dropping the Master Device from tempdb Segments

Once you allocate a second device to `tempdb`, you can drop the master device from the `default`, `system`, and `logsegment` segments. This way, you can be sure that the worktables and other temporary tables in `tempdb` do not contend with other uses on the master device.

Context

By default, the `system`, `default`, and `logsegment` segments for `tempdb` include its 4MB allocation on the master device.

Procedure

1. Alter `tempdb` onto another device, if you have not already done so. For example:

```
alter database tempdb on tune3 = 20
```

2. Issue a `use tempdb` command, and then drop the master device from the segments:

```
sp_dropsegment "default", tempdb, master
```

```
sp_dropsegment "system", tempdb, master
```

```
sp_dropsegment "logsegment", tempdb, master
```

3. To verify the segments no longer include the master device, issue this command against the master database:

```
select dbid, name, segmap
from sysusages, sysdevices
where sysdevices.vdevno= sysusages.vdevno
```

```
and dbid = 2
and (status&2=2 or status&3=3))
```

The `segmap` column should report "0" for any allocations on the master device, indicating that no segment allocations exist:

dbid	name	segmap
2	master	0
2	tune3	7

Alternatively, issue:

```
use tempdb
sp_helpdb 'tempdb'
```

device_fragments	size	usage	created	free kbytes
master	4.0 MB	data only	Feb 7 2008 2:18AM	2376
tune3	20.0 MB	data and log	May 16 2008 1:55PM	16212
device	segment			
master	-- unused by any segments --			
tune3		default		
tune3		logsegment		
tune3		system		

6.8.2 Configure User-Created Temporary Databases

Applications have individual resource and space requirements for temporary databases. Unless you understand your applications requirements, and maintain application to database or group bindings that satisfy these database requirements, make all temporary databases the same size.

If all temporary databases are the same size, applications should not run out of resources or space, regardless of which database is assigned an application or session.

6.8.2.1 Cache User Temporary Databases

Generally, configure caches similarly across temporary databases within a group. The query processor may choose a query plan based on these caching characteristics, and you may see poor performance if the plan is executed using a cache with a different configuration.

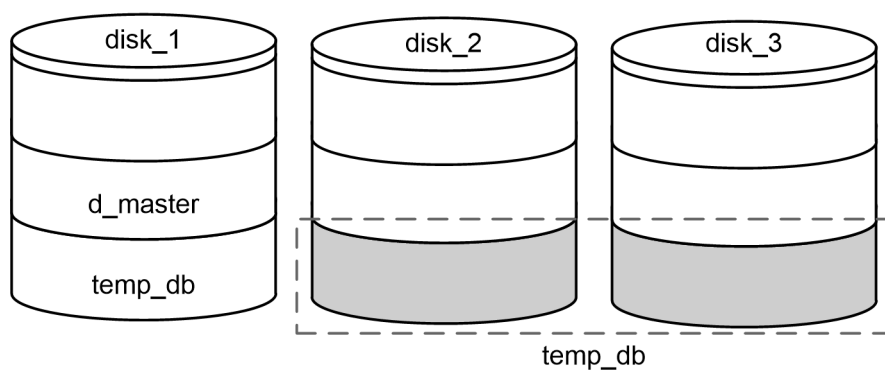
6.8.3 General Guidelines

General guidelines for configuring the temporary databases, which apply to both system and user temporary databases.

6.8.3.1 Multiple Disks for Parallel Query Performance

If temporary databases span multiple devices, you can take advantage of parallel query performance for some temporary tables or worktables

tempdb spanning disks



6.8.3.2 Bind tempdb to its Own Cache

Under normal SAP ASE use, temporary databases make heavy use of the data cache as temporary tables are created, populated, and dropped.

Assigning a temporary database to its own data cache:

- Keeps the activity on temporary objects from flushing other objects out of the default data cache
- Helps spread I/O between multiple caches

6.8.3.2.1 Commands for Cache Binding

Use `sp_cacheconfig` and `sp_poolconfig` to create named data caches and to configure pools of a given size for large I/O. Only a system administrator can configure caches and pools.

i Note

Reference to large I/Os are on a 2K logical page size server. If you have an 8K page size server, the basic unit for the I/O is 8K. If you have a 16K page size server, the basic unit for the I/O is 16K.

For instructions on configuring named caches and pools, see Chapter 4, “Configuring Data Caches” in the *System Administration Guide: Volume 2*.

Once the caches have been configured, and the server has been restarted, you can bind `tempdb` to the new cache:

```
sp_bindcache "tempdb_cache", tempdb
```

6.8.3.3 Determine the Size of Temporary Databases

Allocate sufficient space to temporary databases to handle processes for every concurrent SAP ASE user.

These process include:

- Worktables for merge joins
- Worktables that are created for `distinct`, `group by`, and `order by`, for reformatting, and for the `or` strategy, and for materializing some views and subqueries
- Hashed temporary tables (those created with “#” as the first character of their names)
- Indexes on temporary tables
- Regular user tables in temporary databases
- Procedures built by dynamic SQL

Some applications may perform better if you use temporary tables to split up multitable joins. This strategy is often used for:

- Cases where the optimizer does not choose a good query plan for a query that joins more than four tables
- Queries that join a very large number of tables
- Very complex queries
- Applications that need to filter data as an intermediate step

You might also use temporary databases to:

- Denormalize several tables into a few temporary tables
- Normalize a denormalized table to do aggregate processing

Determine the sizes of temporary databases based on usage scenarios. For most applications, make temporary databases 20 – 25% of the size of your user databases to provide enough space for these uses.

6.8.3.4 Minimize Logging in Temporary Databases

Even though the `trunc log on checkpoint` database option is turned on in temporary databases, SAP ASE still writes changes to temporary databases to the transaction log.

You can reduce log activity in a temporary database by:

- Using `select into` instead of `create table` and `insert`
- Selecting only the columns you need into the temporary tables

6.8.3.4.1 Use select into

When you create and populate temporary tables in a temporary database, use the `select into` command, rather than `create table` and `insert...select`, whenever possible. The `select into/bulkcopy` database option is turned on by default in temporary databases to enable this behavior.

`select into` operations are faster because they are only minimally logged. Only the allocation of data pages is tracked, not the actual changes for each data row. Each data insert in an `insert...select` query is fully logged, resulting in more overhead.

6.8.3.4.2 Use Shorter Rows

If the application creating tables in a temporary database uses only a few columns of a table, you can minimize the number and size of log records.

You can do this by:

- Selecting only the columns you need for the application, rather than using `select *` in queries that insert data into the tables
- Limiting the rows selected to just the rows that the applications requires

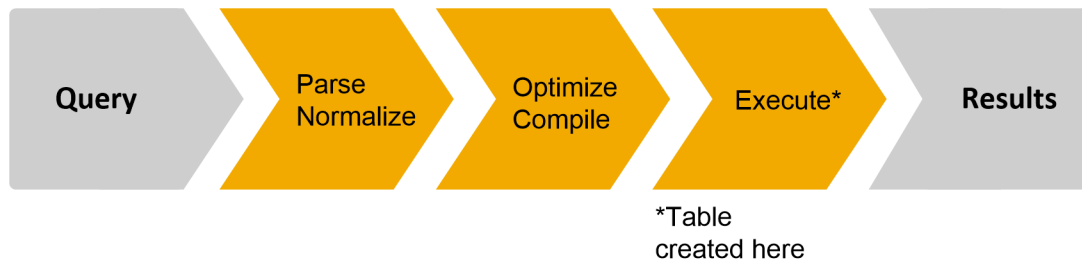
These suggestions also keep the size of the tables themselves smaller.

6.8.3.5 Optimize Temporary Tables

Many uses of temporary tables are simple and brief and require little optimization. However, if your applications require multiple accesses to tables in a temporary database, examine them for possible optimization strategies.

Usually, this involves splitting out the creation and indexing of the table from the access to it by using more than one procedure or batch.

When you create a table in the same stored procedure or batch where it is used, the query optimizer cannot determine how large the table is because the table was not created when the query was optimized. This applies to both temporary tables and regular user tables.



The optimizer assumes that any such table has 10 data pages and 100 rows. If the table is really large, this assumption can lead the optimizer to choose a suboptimal query plan.

These two techniques can improve the optimization of temporary tables:

- Creating indexes on temporary tables
- Breaking complex use of temporary tables into multiple batches or procedures to provide information for the optimizer

6.8.3.5.1 Create Indexes on Temporary Tables

You can define indexes on temporary tables. In many cases, these indexes can improve the performance of queries that use temporary databases.

The optimizer uses these indexes just like indexes on ordinary user tables. The only requirements are:

- The table must contain data when the index is created. If you create the temporary table and create the index on an empty table, SAP ASE does not create column statistics such as histograms and densities. If you insert data rows after creating the index, the optimizer has incomplete statistics.
- The index must exist while the query using it is optimized. You cannot create an index and then use it in a query in the same batch or procedure. The query processor uses indexes created in a stored procedure in queries that are run inside the stored procedure.
- The optimizer may choose a suboptimal plan if rows have been added or deleted since the index was created or since `update statistics` was run.

Providing an index for the optimizer can greatly increase performance, especially in complex procedures that create temporary tables and then perform numerous operations on them.

6.8.3.5.2 Creating Nested Procedures with Temporary Tables

You cannot create `base_proc` until `select_proc` exists, and you cannot create `select_proc` until the temporary table exists.

Procedure

1. Create the temporary table outside the procedure. It can be empty; it just must exist and have columns that are compatible with `select_proc`:

```
select * into #huge_result from ... where 1 = 2
```

2. Create the procedure `select_proc`, as shown above.
3. Drop `#huge_result`.
4. Create the procedure `base_proc`.

6.8.3.5.3 Break tempdb Uses into Multiple Procedures

You can increase performance by using multiple procedures.

For example, this query causes optimization problems with `#huge_result`:

```
create proc base_proc
as
    select *
        into #huge_result
        from ...
    select *
        from tab,
        #huge_result where ...
```

When the `base_proc` procedure calls the `select_proc` procedure, the optimizer can determine the size of the table:

```
create proc select_proc
as
    select *
        from tab, #huge_result where ...
```

```
create proc base_proc
as
    select *
        into #huge_result
        from ...
    exec select_proc
```

If the processing for `#huge_result` requires multiple accesses, joins, or other processes (such as looping with `while`), creating an index on `#huge_result` may improve performance. Create the index in `base_proc` so that it is available when `select_proc` is optimized.

6.9 Log Optimizations for Temporary Databases

SAP ASE does not recover temporary databases when you shut it down or it fails, but SAP ASE does create the temporary databases when you restart the server.

Because temporary databases do not require recovery, SAP ASE optimizes the logging mechanism for temporary databases to improve performance by:

- Single log records – force SAP ASE to flush `syslogs` to disk immediately after SAP ASE logs the record. SAP ASE creates single log records while modifying OAM pages or allocation pages (in a database that is configured to use mixed log and data on the same device). SAP ASE must flush `syslogs` to avoid undetected deadlocks created during buffer pinning. Because SAP ASE does not pin buffers for temporary databases, it need not flush the `syslogs` data for the temporary database when it writes an single log records, which reduces log semaphore contention.
- Flushing dirty pages to disk – for databases that require recovery, SAP ASE flushes dirty pages to disk during the checkpoint, ensuring that, if SAP ASE fails, all committed data is saved to disk. For temporary databases, SAP ASE supports runtime rollbacks, but not failure recovery, allowing it to avoid flushing dirty data pages at the checkpoint.
- Avoiding write-ahead logging – write-ahead logging guarantees that SAP ASE can recover data for committed transactions by reissuing the transactions listed in the log, and undoing the changes performed by aborted or rolled back transactions. SAP ASE does not support write-ahead logging on databases that do not require recovery. Because SAP ASE does not recover temporary database, buffers for temporary databases are not pinned, which allows SAP ASE to skip flushing the temporary database log when it commits a transaction using a temporary database.

6.9.1 User Log Cache (ULC)

SAP ASE contains a separate user log cache (ULC) for the temporary database assigned to the session.

The ULC allows SAP ASE to avoid log flushes when users switch between a user database and session's temporary database, or if all the following conditions are met:

- SAP ASE is currently committing the transaction.
- All the log records are in the ULC.
- There are no post-commit log records.



The configuration option, `session tempdb log cache size`, which allows you to configure the size of the ULC, helps determine how often it needs flushing. See *Reference Manual: Configuration Parameters*.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.