System Administration Guide: Volume 2

SAP® Adaptive Server® Enterprise 16.0

DOCUMENT ID: DC31644-01-1600-01
LAST REVISED: May 2014

Contents

CHAPTER 1 Limiting Access to Server Resources

System administration includes using resource limits to restrict the I/O cost, row count, processing time, or `tempdb` space that an individual login or application can use during critical times, and creating named time ranges to specify contiguous blocks of time for resource limits.

Resource Limits

A resource limit is a set of parameters specified by a system administrator, that prevents queries and transactions from individual logins or applications from monopolizing server resources.

Resource limits are bound to time ranges, which allows the system administrator to define the times during which limits are enforced. When the system administrator modifies a resource limit, all users logged in see the change, including the system administrator

The set of parameters for a resource limit includes the time of day to enforce the limit and the type of action to take. For example, you can prevent huge reports from running during critical times of the day, or kill a session in which a query produces unwanted *Cartesian products*.

Plan Resource Limits

There are a number of issues you should consider when planning resource limits.

For example:

- The times of day and days of the week during which to impose the limit.
- Which users and applications to monitor
- The type of limit to impose:
  - I/O cost (estimated or actual) for queries that may require large numbers of logical and physical reads
  - Row count for queries that may return large result sets
  - Elapsed time for queries that may take a long time to complete, either because of their own complexity or because of external factors such as server load
- Whether to apply a limit to individual queries or to specify a broader scope (query batch or transaction)

- The maximum amount of idle time for users who start a connection but leave it idle for a long time, potentially using system resources such as locks.
- Whether to enforce the I/O cost limits prior to or during execution
- What action to take when the limit is exceeded (issue a warning, abort the query batch or transaction, or kill the session)

Enable Resource Limits

Use the **allow resource limits** configuration parameter to enable resource limits.

A value of 1 enables resource limits; a value of 0 disables them. **allow resource limits** is static, so you must restart the server to reset the changes.

allow resource limits signals the server to allocate internal memory for time ranges, resource limits, and internal server alarms. It also internally assigns applicable ranges and limits to login sessions.

Setting **allow resource limits** to 1 also changes the output of **showplan** and **statistics i/o**:

- **showplan** displays the optimizer's cost estimate for the entire query as a unitless number. This cost estimate is dependent on the table statistics (number and distribution of values) and the size of the appropriate buffer pools. It is independent of such factors as the state of the buffer pools and the number of active users. See *Performance and Tuning Series: Query Processing and Abstract Plans > Using showplan*.
- **statistics i/o** includes the actual total I/O cost of a statement according to the optimizer's costing formula. This value represents the sum of the number of logical I/Os multiplied by the cost of a logical I/O and the number of physical I/Os multiplied by the cost of a physical I/O.

Define Time Ranges

A time range is a contiguous block of time across one or more contiguous days of the week.

SAP® Adaptive Server® Enterprise includes a predefined "at all times" range, which covers the period midnight through midnight, Monday through Sunday. You can create, modify, and drop additional time ranges as necessary for resource limits.

Named time ranges can overlap. However, the limits for a particular user/application combination cannot be associated with named time ranges that overlap.

For example, assume that you limit "joe_user" to returning 100 rows when he is running the payroll application during business hours. Later, you attempt to limit his row retrieval during peak hours, which overlap with business hours. The new limit fails, because it overlaps an existing limit.

You can create different limits that share the same time range. For example, you can put a second limit on "joe_user" during the same time range as the row retrieval limit. For example,

you can limit the amount of time one of his queries can run to the same time range that you used to limit his row retrieval.

When you create a named time range, SAP® ASE stores it in the `systimeranges` system table. Each time range has a range ID number. The "at all times" range is range ID 1. SAP ASE messages refer to specific time ranges.

Determe the Time Ranges You Need

Use a chart to determine the time ranges to create for each server. Monitor server usage throughout the week; then indicate the periods when your server is especially busy or is performing crucial tasks that should not be interrupted.

Day	Time	00:000	01:000	02:000	03:000	04:000	05:000	06:000	07:000	08:000	09:000	10:000	11:000	12:000	13:000	14:000	15:000	16:000	17:000	18:000	19:000	20:000	21:000	22:000	23:000	00:000
Mon																										
Tues																										
Wed																										
Thur																										
Fri																										
Sat																										
Sun																										

Use **sp_add_time_range** to:

- Name the time range
- Specify the days of the week to begin and end the time range
- Specify the times of the day to begin and end the time range

See **sp_add_time_range** in the *Reference Manual: Procedures*.

A Time Range Example

This example assumes that two critical jobs run every week.

- Job 1 runs from 07:00 to 10:00 on Tuesday and Wednesday.
- Job 2 runs from 08:00 on Saturday to 13:00 on Sunday.

The following table uses "1" to indicate when job 1 runs and "2" to indicate when job 2 runs:

Day	Time	00:000	01:000	02:000	03:000	04:000	05:000	06:000	07:000	08:000	09:000	10:000	11:000	12:000	13:000	14:000	15:000	16:000	17:000	18:000	19:000	20:000	21:000	22:000	23:000	00:000
Mon																										
Tues									1	1	1	1														
Wed									1	1	1	1														
Thurs																										
Fri																										
Sat										2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Sun		2	2	2	2	2	2	2	2	2	2	2	2	2	2											

Job 1 can be covered by a single time range, `tu_wed_7_10`:

```
sp_add_time_range tu_wed_7_10, tuesday, wednesday, "7:00", "10:00"
```

Job 2, however, requires two separate time ranges, for Saturday and Sunday:

```
sp_add_time_range saturday_night, saturday, saturday, "08:00",
"23:59"
sp_add_time_range sunday_morning, sunday, sunday, "00:00", "13:00"
```

Modifying a Named Time Range

Use **sp_modify_time_range** to modify time ranges, including changes to the days of the week, or times of the day.

See **sp_modify_time_range** in the *Reference Manual: Procedures*.

Note: You cannot modify the "at all times" time range.

For example, to change the end day of the *business_hours* time range to Saturday, retaining the existing start day, start time, and end time, enter:

```
sp_modify_time_range business_hours, NULL, Saturday, NULL, NULL
```

To specify a new end day and end time for the *before_hours* time range, enter:

```
sp_modify_time_range before_hours, NULL, Saturday, NULL, "08:00"
```

Dropping a Named Time Range

Use **sp_drop_time_range** to drop a user-defined time range.

Note: You cannot drop the "at all times" time range or any time range for which resource limits are defined.

To remove the *evenings* time range from the `systimeranges` system table in the `master` database, enter:

```
sp_drop_time_range evenings
```

See **sp_drop_time_range** in the *Reference Manual: Procedures*.

When Do Time Range Changes Take Effect?

Active time ranges are bound to a login session at the beginning of each query batch. A change in the server's active time ranges due to a change in actual time has no effect during the processing of a query batch.

In other words, if a resource limit restricts query batches during a given time range, but the query batch begins before that time range becomes active, the query batch that is already running is not affected by the resource limit. However, if you run a second query batch during the same login session, that query batch is affected by the change in time.

Adding, modifying, and deleting time ranges does not affect the active time ranges for the login sessions currently in progress.

If a resource limit has a transaction as its scope, and a change occurs in the server's active time ranges while a transaction is running, the newly active time range does not affect the transaction currently in progress.

Identify Users and Limits

For each resource limit, specify the object to which the limit applies, for example, all applications used by a particular login, all logins that use a particular application, or a specific application used by a particular login.

An *application* is a client program that is running on top of SAP ASE, accessed through a particular login. To run an application on SAP ASE, specify its name through the CS_APPNAME connection property using **cs_config** (an Open Client™ Client-Library™ application) or the DBSETLAPP function in Open Client DB-Library™. To list named applications running on your server, select the `program_name` column from the `master..sysprocesses` table.

For more information about the CS_APPNAME connection property, see the *Open Client Client-Library/C Reference Manual*. For more information on the DBSETLAPP function, see the *Open Client DB-Library/C Reference Manual*.

See also

- *Choosing a Limit Type* on page 7

Identifying Heavy-Usage Users

Before you implement resource limits, run **sp_reportstats**. The output from this procedure can help you identify users with heavy system usage.
For example:

```
sp_reportstats
```

```
Name    Since          CPU     Percent CPU  I/O     Percent I/O
------  -----------    -----   ------------ -----   -------------
probe   jun 19 2007    0       0%           0       0%
julie   jun 19 2007    10000   24.9962%     5000    24.325%
jason   jun 19 2007    10002   25.0013%     5321    25.8866%
ken     jun 19 2007    10001   24.9987%     5123    24.9234%
kathy   jun 19 2007    10003   25.0038%     5111    24.865%
```

```
Total CPU   Total I/O
---------   ---------
40006       20555
```

The I/O and percent I/O columns indicate indicates that usage is balanced among the users. For more information about chargeback accounting, see *System Administration Guide: Volume 1 > Setting Configuration Paramters*.

Identifying Heavy-Usage Applications

To identify the applications running on your system and the users who are running them, query the `sysprocesses` system table in the `master` database.

This query determines that **isql**, **payroll**, **perl**, and **acctng** are the only client programs whose names were passed to the SAP ASE:

```
select spid, cpu, physical_io,
  substring(user_name(uid),1,10) user_name,
  hostname, program_name, cmd
from sysprocesses
```

```
spid  cpu    physical_io  user_name hostname program_name cmd
----  ---    -----------  --------- -------- ------------ ------
  17    4          12748  dbo       sabrina  isql         SELECT
 424    5              0  dbo       HOWELL   isql         UPDATE
 526    0            365  joe       scotty   payroll      UPDATE
 568    1           8160  dbo       smokey   perl         SELECT
 595   10              1  dbo       froth    isql         DELETE
 646    1              0  guest     walker   isql         SELECT
 775    4          48723  joe_user  mohindra acctng       SELECT

(7 rows affected)
```

Because `sysprocesses` is built dynamically to report current processes, repeated queries produce different results. Repeat this query throughout the day over a period of time to determine which applications are running on your system.

The CPU and physical I/O values are periodically flushed to the `syslogins` system table, where they increment the values shown by **sp_reportstats**.

After identifying the applications running on your system, use **showplan** and **statistics io** to evaluate the resource usage of the queries in the applications.

Choosing a Limit Type

After you determine the users and applications to limit, chose the resource limit type. This table describes the function and scope of each limit type and indicates the tools that help determine whether a particular query might benefit from this type of limit. You may want to create more than one type of limit for a given user and application.

Limit Type	Use for Queries That	Measuring Resource Usage	Scope	Enforced During
io_cost	Require many logical and physical reads.	Use **set showplan on** before running the query, to display its estimated I/O cost; use **set statistics io on** to observe the actual I/O cost.	Query	Preexecution or execution
row_count	Return large result sets.	Use the *@@rowcount* global variable to help develop appropriate limits for row count.	Query	Execution
elapsed_time	Take a long time to complete, either because of their own complexity or because of external factors such as server load or waiting for a lock.	Use **set statistics time on** before running the query, to display elapsed time in milliseconds.	Query batch or transaction	Execution
tempdb_space	Use all space in `tempdb` when creating work or temporary tables.	Number of pages used in `tempdb` per session.	Query batch or transaction	Execution
idle_time	Are inactive.	Time, in seconds, during which the connection is inactive.	Individual processes	Preexecution

The `spt_limit_types` system table stores information about each limit type.

See also

- *Identify Users and Limits* on page 5

Determining Time of Enforcement

Time of enforcement is the phase of query processing during which SAP ASE applies a given resource limit.

Resource limits occur during:

- Preexecution – SAP ASE applies resource limits prior to execution, based on the optimizer's I/O cost estimate. Use this type of limit to prevent execution of potentially expensive queries. I/O cost is the only resource type that can be limited at preexecution time.

  When evaluating the I/O cost of data manipulation language (DML) statements within the clauses of a conditional statement, SAP ASE individually considers each DML statement. It evaluates all statements, even though only one clause is actually executed.

  A preexecution time resource limit can have only a query limit scope; that is, the values of the resources being limited at compile time are computed and monitored only on a query-by-query basis.

  SAP ASE does not enforce preexecution time resource limits statements in a trigger.
- Execution – SAP ASE applies resource limits at runtime, and is usually used to prevent a query from monopolizing server and operating system resources. Execution time limits may use more resources (additional CPU time as well as I/O) than preexecution time limits.

Determining the Scope of Resource Limits

The *scope* parameter (for example, **sp_add_resource_limit** *scope* or **sp_help_resource_limit** *scope*) specifies the duration of a limit in Transact-SQL statements.

The possible limit scopes are query, query batch, and transaction:

- Query – SAP ASE applies resource limits to any single Transact-SQL statement that accesses the server; for example, **select**, **insert**, and **update**. When you issue these statements within a query batch, SAP ASE evaluates them individually.

  SAP ASE considers a stored procedure to be a series of DML statements. It evaluates the resource limit of each statement within the stored procedure. If a stored procedure executes another stored procedure, SAP ASE evaluates each DML statement within the nested stored procedure at the inner nesting level.

  SAP ASE checks preexecution time resource limits with a query scope, one nesting level at a time. As SAP ASE enters each nesting level, it checks the active resource limits against the estimated resource usage of each DML statement prior to executing any of the statements at that nesting level. A resource limit violation occurs if the estimated resource usage of any DML query at that nesting level exceeds the limit value of an active resource limit. SAP ASE takes the action that is bound to the violated resource limit.

  SAP ASE checks execution-time resource limits with a query scope against the cumulative resource usage of each DML query. A limit violation occurs when the resource usage of a

query exceeds the limit value of an active execution-time resource limit. Again, SAP ASE takes the action that is bound to that resource limit.

- Query batch – consists of one or more Transact-SQL statements; for example, in **isql**, a group of queries becomes a query batch when executed by a single **go** command terminator.

  The query batch begins at nesting level 0; each call to a stored procedure increments the nesting level by 1 (up to the maximum nesting level). Each return from a stored procedure decrements the nesting level by 1.

  Only execution-time resource limits can have a query batch scope.

  SAP ASE checks execution-time resource limits with a query batch scope against the cumulative resource usage of the statements in each query batch. A limit violation occurs when the resource usage of the query batch exceeds the limit value of an active execution-time resource limit. SAP ASE takes the action that is bound to that resource limit.
- Transaction – SAP ASE applies limits with a transaction scope to all nesting levels during the transaction against the cumulative resource usage for the transaction.

  A limit violation occurs when the resource usage of the transaction exceeds the limit value of an active execution-time resource limit. SAP ASE takes the action that is bound to that resource limit.

  Only execution-time resource limits can have a transaction scope.

  SAP ASE does not recognize nested transactions when applying resource limits. A resource limit on a transaction begins when *@@trancount* is set to 1 and ends when *@@trancount* is set to 0.
- Session – idle time limits are applied to the sessions in which the limit is active.

See also

- *Scope of the io_cost Limit Type* on page 11
- *Scope of the elapsed_time Limit Type* on page 11
- *Scope of the row_count Limit Type* on page 12

Understanding Limit Types

Resource limits allow you to limit resource usage in different ways, including by I/O cost, elapsed time, row count, tempdb space usage, and idle time.

Limiting I/O Cost

I/O cost is based on the number of logical and physical accesses (“reads”) used during query processing. To determine the most efficient processing plan before execution, the SAP ASE optimizer uses both logical and physical resources to compute an estimated I/O cost.

SAP ASE uses the result of the optimizer’s costing formula as a unitless number; that is, a value not necessarily based on a single unit of measurement, such as seconds or milliseconds.

To set resource limits, you must understand how those limits translate into runtime system overhead. For example, you must know the effect that a query with a cost of *x* logical and of *y* physical I/Os has on a production server.

Limiting **io_cost** can control I/O-intensive queries, including queries that return a large result set. However, if you run a simple query that returns all the rows of a large table, and you do not have current statistics on the table's size, the optimizer may not estimate that the query exceeds the **io_cost** resource limit. To prevent queries from returning large result sets, create a resource limit on **row_count**.

The tracking of I/O cost limits may be less precise for partitioned tables than for unpartitioned tables when SAP ASE is configured for parallel query processing. See *Performance and Tuning Series: Query Processing and Abstract Plans > Parallel Query Processing*.

Identify I/O Costs

To develop appropriate limits for I/O cost, use **set** commands to determine the number of logical and physical reads required for some typical queries.

- **set showplan on** displays the optimizer's cost estimate. Use this information to set preexecution time resource limits. A preexecution time resource limit violation occurs when the optimizer's I/O cost estimate for a query exceeds the limit value. Such limits prevent the execution of potentially expensive queries.
- **set statistics io on** displays the number of actual logical and physical reads required. Use this information to set execution-time resource limits. An execution-time resource limit violation occurs when the actual I/O cost for a query exceeds the limit value.

Statistics for actual I/O cost include access costs only for user tables and worktables involved in the query. SAP ASE may use other tables internally; for example, it accesses `sysmessages` to print out statistics. Therefore, there may be instances when a query exceeds its actual I/O cost limit, even though the statistics indicate otherwise.

In costing a query, the optimizer assumes that every page needed requires a physical I/O for the first access and is found in the cache for repeated accesses. Actual I/O costs may differ from the optimizer's estimated costs, for several reasons.

The estimated cost is higher than the actual cost if some pages are already in the cache or if the statistics are incorrect. The estimated cost may be lower than the actual cost if the optimizer chooses 16K I/O, and some of the pages are in 2K cache pools, which require many 2K I/Os. Also, if a big join forces the cache to flush its pages back to disk, repeated access may require repeated physical I/Os.

The optimizer's estimates are inaccurate if the distribution or density statistics are out of date or cannot be used.

Calculate the I/O Cost of a Cursor

The cost estimate for processing a cursor is calculated at **declare cursor** time for all cursors except execute cursors, which is calculated when the cursor opens.

Preexecution resource limits on I/O cost are enforced at **open** *cursorname* time for all cursor types. The optimizer recalculates the limit value each time the user attempts to open the cursor.

An execution-time resource limit applies to the cumulative I/O cost of a cursor from the time the cursor opens to the time it closes. The optimizer recalculates the I/O limit each time a cursor opens.

See *Transact-SQL Users Guide > Cursors: Accessing Data*.

Scope of the io_cost Limit Type

A resource limit that restricts I/O cost applies only to single queries. If you issue several statements in a query batch, SAP ASE evaluates the I/O usage for each query.

See also

- *Determining the Scope of Resource Limits* on page 8

Limiting Elapsed Time

Elapsed time is the number of seconds required to execute a query batch or transaction, determined by such factors as query complexity, server load, and waiting for locks.

To develop appropriate limits for elapsed time, use information you have gathered with **set statistics time**. You can limit the elapsed-time resource only at execution.

With **set statistics time** set **on**, run some typical queries to determine processing time in milliseconds. Convert milliseconds to seconds when you create the resource limit.

Elapsed-time resource limits are applied to all SQL statements in the limit's scope (query batch or transaction), not only to the DML statements. A resource limit violation occurs when the elapsed time for the appropriate scope exceeds the limit value.

Separate elapsed time limits are not applied to nested stored procedures or transactions. In other words, if one transaction is nested within another, the elapsed time limit applies to the outer transaction, which encompasses the elapsed time of the inner transaction. Therefore, if you are counting the running time of a transaction, it includes the running time for all nested transactions.

Scope of the elapsed_time Limit Type

The scope of a resource limit that restricts elapsed time is either a query batch or transaction.

See also

- *Determining the Scope of Resource Limits* on page 8

Limiting the Size of the Result Set

The **row_count** limit type limits the number of rows returned to the user. A limit violation occurs when the number of rows returned by a **select** statement exceeds the limit value. If the resource limit issues a warning as its action, and a query exceeds the row limit, the full number of rows are returned, followed by a warning that indicates the limit value; for example:

```
Row count exceeded limit of 50.
```

If the resource limit's action aborts the query batch or transaction or kills the session, and a query exceeds the row limit, only the limited number of rows are returned and the query batch, transaction, or session aborts. SAP ASE displays a message similar to:

```
Row count exceeded limit of 50.
Transaction has been aborted.
```

The **row_count** limit type applies to all **select** statements at execution. You cannot limit an estimated number of rows returned at preexecution time.

Use the *@@rowcount* global variable to help develop appropriate limits for row count. Selecting this variable after running a typical query can tell you how many rows the query returned.

A row count limit applies to the cumulative number of rows that are returned through a cursor from the time the cursor opens to the time it closes. The optimizer recalculates the **row_count** limit each time a cursor opens.

Scope of the row_count Limit Type

A resource limit that restricts row count applies only to single queries, not to cumulative rows returned by a query batch or transaction.

See also

- *Determining the Scope of Resource Limits* on page 8

Setting Limits for tempdb Space Usage

The **tempdb_space** resource limit restricts the number of pages a `tempdb` database can have during a single session. If a user exceeds the specified limit, the session can be terminated, or the batch or transaction aborted.

For queries executed in parallel, the **tempdb_space** resource limit is distributed equally among the parallel threads. For example, if the **tempdb_space** resource limit is set at 1500 pages and a user executes the following with three-way parallelism, each parallel thread can create a maximum of 500 pages in `tempdb`:

```
select into #temptable from partitioned_table
```

The system administrator or database administrator sets the **tempdb_space** limit using **sp_add_resource_limit**, and drops the **tempdb_space** limit using **sp_drop_resource_limit**.

Limiting Idle Time

Idle time is the number of seconds that a connection remains inactive, waiting for user input. An inactive connection continues to use server resources, and may also hold resources (for example, locks) that can block other active processes running on the same server.

idle_time allows you to set time limits for idle connections. If a connection is idle beyond the limit set, SAP ASE stops the process running the connection or issues a warning.

The syntax is:

```
sp_add_resource_limit user, application, time_range, idle_time ,
kill_time, enforcement_time, action, scope
```

This creates a new limit for user "sa" for **isql** connections:

```
sp_add_resource_limit sa, isql, 'at all times', idle_time, 10,2,4,8
```

See the *Reference Manual: Procedures*.

Creating a Resource Limit

Create a new resource limit using **sp_add_resource_limit**.

The syntax is:

```
sp_add_resource_limit name, appname, rangename, limittype,
limit_value, enforced, action, scope
```

See **sp_add_resource_limit** in the *Reference Manual: Procedures*.

Resource Limit Examples

The following are examples of setting resource limits.

- Example 1– This example creates a resource limit that applies to all users of the **payroll** application because the name parameter is NULL.

  ```
  sp_add_resource_limit NULL, payroll, tu_wed_7_10, elapsed_time,
  120, 2, 1, 2
  ```

  The limit is valid during the `tu_wed_7_10` time range. The limit type, **elapsed_time**, is set to a value of 120 seconds. Because **elapsed_time** is enforced only at execution time, the *enforced* parameter is set to 2. The *action* parameter is set to 1, which issues a warning. The limit's *scope* is set to 2, query batch, by the last parameter. Therefore, when the elapsed time of the query batch takes more than 120 seconds to execute, SAP ASE issues a warning.
- Example 2 – This example creates a resource limit that applies to all ad hoc queries and applications run by "joe_user" during the `saturday_night` time range:

  ```
  sp_add_resource_limit joe_user, NULL, saturday_night,
      row_count, 5000, 2, 3, 1
  ```

If a query (*scope* = 1) returns more than 5000 rows, SAP ASE aborts the transaction (*action* = 3). This resource limit is enforced at execution time (*enforced* = 2).

- Example 3 – This example also creates a resource limit that applies to all ad hoc queries and applications run by "joe_user:"

```
sp_add_resource_limit joe_user, NULL, "at all times",
    io_cost, 650, 1, 3, 1
```

However, this resource limit specifies the default time range, "at all times." When the optimizer estimates that the **io_cost** of the query (*scope* = 1) would exceed the specified value of 650, SAP ASE aborts the transaction (*action* = 3). This resource limit is enforced at preexecution time (*enforced* = 1).

Note: Although SAP ASE terminates the current transaction when it reaches its time limit, you receive no 1105 error message until you issue another SQL command or batch; in other words, the message appears only when you attempt to use the connection again.

Getting Information on Existing Limits

Use **sp_help_resource_limit** to get information about existing resource limits.

When you use **sp_help_resource_limit** without any parameters, SAP ASE lists all resource limits within the server.

For example:

```
sp_help_resource_limit
```

```
name     appname rangename rangeid limitid limitvalue enforced  acti
on scope
----     ------- --------- ------- ------- ---------- --------  ----
-- -----
NULL     acctng  evenings       4       2        120         2       1     2
stein    NULL    weekends       1       3       5000         2       1     1
joe_user acctng  bus_hours       5       3       2500         2
  2     1
joe_user finance bus_hours       5       2        160         2
  1     6
wong     NULL    mornings       2       3       2000         2       1     1
wong     acctng  bus_hours       5       1         75         1       3     1
```

The `rangeid` column prints the value from `systimeranges.id` that corresponds to the name in the `rangename` column. The `limitvalue` column reports the value set by **sp_add_resource_limit** or **sp_modify_resource_limit**. This table shows the meaning of the values in the `limitid`, `enforced`, `action`, and `scope` columns.

Table 1. Values for sp_help_resource_limit output

Column	Meaning	Value
limitid	What kind of limit is it?	1– I/O cost 2 – elapsed time 3 – row count
enforced	When is the limit enforced?	1 – before execution 2 – during execution 3 – both
action	What action is taken when the limit is hit?	1– issue a warning 2 – abort the query batch 3 – abort the transaction 4 – kill the session
scope	What is the scope of the limit?	1 – query 2 – query batch 4 – transaction 6 – query batch + transaction

If a system administrator specifies a login name when executing **sp_help_resource_limit**, SAP ASE lists all resource limits for that login. The output displays not only resource limits specific to the named user, but all resource limits that pertain to all users of specified applications, because the named user is included among all users.

For example, the following output shows all resource limits that apply to "joe_user." Because a resource limit is defined for all users of the **acctng** application, this limit is included in the output.

```
sp_help_resource_limit joe_user
```

```
name     appname rangename rangeid limitid limitvalue enforced  acti
on scope
----     ------- --------- ------- ------- ---------- --------  ----
-- -----
NULL    acctng  evenings       4       2       120         2       1     2
joe_user acctng  bus_hours        5        3       2500          2
  2     1
joe_user
finance bus_hours        5        2        160          2        1      6
```

See **sp_help_resource_limit** in the *Reference Manual: Procedures*.

Modifying Resource Limits

Use **sp_modify_resource_limit** to specify a new limit value or a new action to take when the limit is exceeded or both. You cannot change the login or application to which a limit applies or specify a new time range, limit type, enforcement time, or scope.
The syntax of **sp_modify_resource_limit** is:

```
sp_modify_resource_limit name, appname, rangename, limittype,
    limitvalue, enforced, action, scope
```

See **sp_modify_resource_limit** in the *Reference Manual: Procedures*.

Dropping Resource Limits

Use **sp_drop_resource_limit** to drop a resource limit from an SAP ASE.
The syntax is:

```
sp_drop_resource_limit {name , appname } [, rangename, limittype,
    enforced, action, scope]
```

Specify enough information to uniquely identify the limit. You must specify a non-null value for either *name* or *appname*.

See **sp_drop_resource_limit** in the *Reference Manual: Procedures*.

Resource Limit Precedence

SAP ASE provides precedence rules for time ranges and resource limits.

For each login session during the currently active time ranges, only one limit can be active for each distinct combination of limit type, enforcement time, and scope.

The precedence rules for determining the active limit are as follows:

- If no limit is defined for the login ID for either the “at all times” range or the currently active time ranges, there is no active limit.
- If limits are defined for the login for both the “at all times” and time-specific ranges, the limit for the time-specific range takes precedence.

Since either the user’s login name or the application name, or both, are used to identify a resource limit, SAP ASE observes a predefined search precedence while scanning the `sysresourcelimits` table for applicable limits for a login session.

The precedence of matching ordered pairs of login name and application name is:

Level	Login name	Application name
1	joe_user	**payroll**
2	NULL	**payroll**
3	joe_user	NULL

If one or more matches are found for a given precedence level, no further levels are searched. This prevents conflicts regarding similar limits for different login/application combinations.

If no match is found at any level, no limit is imposed on the session.

CHAPTER 2 Mirroring Database Devices

Disk mirroring can provide nonstop recovery in the event of media failure.

The **disk mirror** command duplicates an SAP ASE database device, that is, all writes to the device are copied to a separate physical device. If one device fails, the other contains an up-to-date copy of all transactions.

When a read or write to a mirrored device fails, SAP ASE "unmirrors" the bad device and displays error messages. SAP ASE continues to run unmirrored.

Determining Which Devices to Mirror

When deciding to mirror a device, you must weigh such factors as the costs of system downtime, possible reduction in performance, and the cost of storage media. Considering these issues can help you decide whether to mirror only the transaction logs, all devices on a server, or selected devices.

Note: You cannot mirror a dump device.

Mirror all default database devices so that you are protected if a **create** or **alter database** command affects a database device in the default list.

In addition to mirroring user database devices, put transaction logs on a separate database device. For even greater protection, mirror the database device used for transaction logs.

To put a database's transaction log (that is, the system table `syslogs`) on a different device than the one on which the rest of the database is stored, name the database device and the log device when you create the database. You can also use **alter database** to add a second device and then run **sp_logdevice**.

When weighing cost and performance trade-offs, consider:

- Speed of recovery – you can achieve nonstop recovery when the `master` and user databases (including logs) are mirrored and can recover without the need to reload transaction logs.
- Storage space – immediate recovery requires full redundancy (all databases and logs mirrored), which consumes disk space.
- Impact on performance – mirroring the user databases (as shown in the image below and in) increases the time needed to write transactions to both disks.

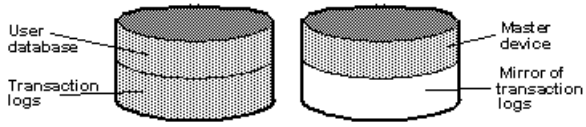
Mirroring Using Minimal Physical Disk Space

The master device and a mirror of the user database transaction log are stored in separate partitions on one physical disk. The other disk stores the user database and its transaction log in two separate disk partitions.

If the disk with the user database fails, you can restore the user database on a new disk from your backups and the mirrored transaction log.

If the disk with the master device fails, you can restore the master device from a database dump of the `master` database and remirror the user database's transaction log.

This figure illustrates the "minimum guaranteed configuration" for database recovery in case of hardware failure.



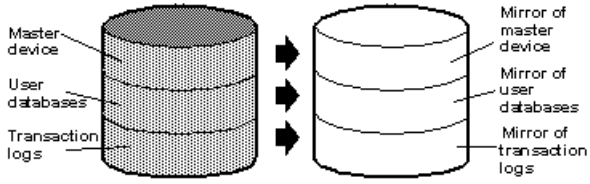
This configuration minimizes the amount of disk storage required, but the mirror of the transaction log ensures full recovery. However, this configuration does not provide nonstop recovery because the `master` and user databases are not being mirrored and must be recovered from backups.

Mirroring for Nonstop Recovery

In some systems, the master device, user databases, and transaction log are all stored on different partitions of the same physical device and are all mirrored to a second physical device.

This configuration provides nonstop recovery from hardware failure. Working copies of the `master` and user databases and log on the primary disk are all mirrored, and failure of either disk does not interrupt SAP ASE users.

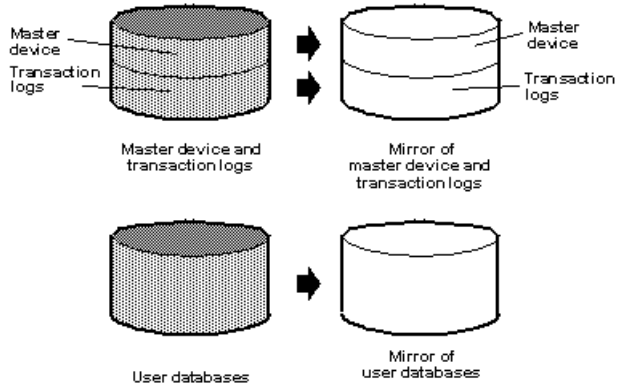
Figure 1: Disk mirroring for Rapid Recovery



With this configuration, all data is written twice, once to the primary disk and once to the mirror. Applications that involve many writes may be slower with disk mirroring than without mirroring.

This figure illustrates another configuration with a high level of redundancy. In this configuration, all three database devices are mirrored, but the configuration uses four disks instead of two. This configuration speeds performance during write transactions because the database transaction log is stored on a different device from the user databases, and the system can access both with less disk head travel.

Figure 2: Disk Mirroring: Keeping Transaction Logs on a Separate Disk



Conditions That Do Not Disable Mirroring

SAP ASE disables a mirror only when it encounters an I/O error on a mirrored device.

For example, if SAP ASE tries to write to a bad block on the disk, the resulting error disables mirroring for the device. However, processing continues without interruption on the unaffected mirror.

Mirroring is not disabled when:

- An unused block on a device is bad. SAP ASE does not detect an I/O error and disables mirroring until it accesses the bad block.
- Data on a device is overwritten. This might happen if a mirrored device is mounted as a UNIX file system, and UNIX overwrites the SAP ASE data. This causes database corruption, but mirroring is not disabled, since SAP ASE does not encounter an I/O error.
- Incorrect data is written to both the primary and secondary devices.
- The file permissions on an active device are changed. Some system administrators may try to test disk mirroring by changing permissions on one device, hoping to trigger I/O failure and unmirror the other device. But the UNIX operating system does not check permissions

on a device after opening it, so the I/O failure does not occur until the next time the device is started.

Disk mirroring is not designed to detect or prevent database corruption. Some of the scenarios described can cause corruption, so you should regularly run consistency checks such as **dbcc checkalloc** and **dbcc checkdb** on all databases.

See also

- *Chapter 12, Checking Database Consistency* on page 221

Disk Mirroring Commands

The **disk mirror**, **disk unmirror**, and **disk remirror** commands control disk mirroring. You can issues any of these commands while the devices are in use. In other words, you need not stop databases to set up or modify disk mirroring.

Note: disk mirror, **disk unmirror**, and **disk remirror** alter the `sysdevices` table in the `master` database. After issuing any of these commands, dump the `master` database to ensure recovery in case `master` is damaged.

Initializing Mirrors

disk mirror starts disk mirroring. Do not use **disk init** to initialize a mirror device. A database device and its mirror constitute one logical device.

The **disk mirror** command adds the mirror name to the `mirrorname` column in the `sysdevices` table.

The **disk mirror** syntax is:

```
disk mirror
     name = "device_name" ,
     mirror = "physicalname"
     [ , writes = { serial | noserial }]
```

Unmirroring a Device

Disk mirroring is automatically deactivated when one of the two physical devices fails.

When a read or write to a mirrored device is unsuccessful, SAP ASE prints error messages. SAP ASE continues to run, unmirrored. You must remirror the disk to restart mirroring.

Use the **disk unmirror** command to stop the mirroring process during hardware maintenance:

```
disk unmirror
     name = "device_name"
     [, side = { "primary" | secondary }]
     [, mode = { retain | remove }]
```

Effects on System Tables

The **mode** parameter of the **disk unmirror** command changes the `status` column in `sysdevices` to indicate that mirroring has been disabled.

See *System Administration Guide: Volume 1 > Initializing Database Devices* and *Reference Manual: Commands*.

Its effects on the `phyname` and `mirrorname` columns in `sysdevices` depend on the **side** argument also:

		side	
		primary	secondary
mode	**remove**	Name in `mirrorname` moved to `phyname` and `mirrorname` set to null; `status` changed	Name in `mirrorname` removed; `status` changed
	retain	Names unchanged; `status` changed to indicate which device is being deactivated	

This example suspends the operation of the primary device:

```
disk unmirror
  name = "tranlog",
  side = "primary"
```

Restarting Mirrors

Use **disk remirror** to restart a mirror process that has been suspended due to a device failure or with **disk unmirror**.
The syntax is:

```
disk remirror
     name = "device_name"
```

This command copies the database device to its mirror.

waitfor mirrorexit

Since disk failure can impair system security, you can include the **waitfor mirrorexit** command in an application to perform specific tasks when a disk becomes unmirrored.
For example:

```
begin
    waitfor mirrorexit
    commands to be executed
end
```

The commands depend on your applications. You may want to add certain warnings in applications that perform updates, or use **sp_dboption** to make certain databases read-only if the disk becomes unmirrored.

Note: SAP ASE knows that a device has become unmirrored only when it attempts I/O to the mirror device. On mirrored databases, this occurs at a checkpoint or when the SAP ASE buffer must be written to disk. On mirrored logs, I/O occurs when a process writes to the log, including any committed transaction that performs data modification, a checkpoint, or a database dump.

waitfor mirrorexit and the error messages that are printed to the console and error log on mirror failure are activated only by these events.

Mirroring the Master Device

If you choose to mirror the device that contains the `master` database, in a UNIX environment, you must edit the runserver file for your SAP ASE so that the mirror device starts when the server starts.
On UNIX, add the `-r` flag and the name of the mirror device:

```
dataserver -d /dev/rsd1f -r /dev/rs0e -e/sybase/install/errorlog
```

For information about mirroring the master device on Windows, see the *Utility Guide*.

Getting Information About Devices and Mirrors

For a report on all SAP ASE devices on your system (user database devices and their mirrors, as well as dump devices), execute **sp_helpdevice**.

See the *Reference Manual: Procedures*.

Disk Mirroring Tutorial

The steps for disk mirrorming include intializing, mirroring, unmirroring, remirroring, disabling, and then removing a test device.

1. Initialize a new test device using:

   ```
   disk init name = "test",
   physname = "/usr/new_user/test.dat",
   size=5120
   ```

   This inserts the following values into columns of `master..sysdevices`:

   ```
   name  phyname              mirrorname          status
   test  /usr/new_user/test.dat   NULL                  16386
   ```

   Status 16386 indicates that the device is a physical device (2, 0x00000002), and any writes are to a UNIX file (16384, 0x00004000). Since the `mirrorname` column is null, mirroring is not enabled on this device.

2. Mirror the test device using:

   ```
   disk mirror name = "test",
   mirror = "/usr/new_user/test.mir"
   ```

This changes the `master..sysdevices` columns to:

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   /usr/new_user/test.mir  17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

3. Disable the mirror device (the secondary side), but retain that mirror:

```
disk unmirror name = "test",
side = secondary, mode = retain
```

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   /usr/new_user/test.mir  18658
```

Status 18658 indicates that the device is mirrored (64, 0x00000040), and the mirror device has been retained (2048, 0x00000800), but mirroring has been disabled (512 bit off), and only the primary device is used (256 bit off). Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file (16384, 0x00004000) and are in serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

4. Remirror the test device:

```
disk remirror name = "test"
```

This resets the `master..sysdevices` columns to:

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   /usr/new_user/test.mir  17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

5. Disable the test device (the primary side), but retain that mirror:

```
disk unmirror name = "test",
side = "primary", mode = retain
```

This changes the `master..sysdevices` columns to:

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   /usr/new_user/test.mir  16866
```

Status 16866 indicates that the device is mirrored (64, 0x00000040), but mirroring has been disabled (512 bit off) and that only the secondary device is used (256, 0x00000100). Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file (16384, 0x00004000), and are in serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

6. Remirror the test device:

```
disk remirror name = "test"
```

This resets the `master..sysdevices` columns to:

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   /usr/new_user/test.mir  17122
```

Status 17122 indicates that mirroring is currently enabled (512, 0x00000200) on this device. Reads are mirrored (128, 0x00000080), and writes are mirrored to a UNIX file device (16384, 0x00004000), the device is mirrored (64, 0x00000040), and serial (32, 0x00000020). The device is a physical disk (2, 0x00000002).

7. Disable the test device (the primary side), and remove that mirror:

```
disk unmirror name = "test", side = "primary",
mode = remove
```

This changes the `master..sysdevices` columns to:

```
name  phyname              mirrorname          status
test  /usr/new_user/test.dat   NULL                    16386
```

Status 16386 indicates that the device is a physical device (2, 0x00000002), and any writes are to a UNIX file (16384, 0x00004000). Since the `mirrorname` column is null, mirroring is not enabled on this device.

8. Remove the test device to complete the tutorial:

```
sp_dropdevice test
```

This removes all entries for the test device from `master..sysdevices`.

Disk Resizing and Mirroring

Use **disk resize** only when mirroring is permanently disabled.

If you try to run **disk resize** on a device that is mirrored, you see:

```
disk resize can proceed only when mirroring is permanently disabled.
Unmirror secondary with mode = 'remove' and re-execute disk resize
command.
```

When mirroring is only temporarily disabled, two scenarios can arise:

- The primary device is active while the secondary device is temporarily disabled and produces the same error as shown above.
- The secondary device is active while the primary device is temporarily disabled and produces an error with this message:

  ```
  disk resize can proceed only when mirroring is permanently
  disabled. Unmirror primary with mode = 'remove' and re-execute the
  command.
  ```

1. Permanently disable mirroring on the device.

2. Increase the size of the primary device.
3. Physically remove the mirror device (in case of file).
4. Reestablish mirroring.

CHAPTER 3 Configuring Memory

All database object pages are sized in terms of the *logical page size*, which you specify when you build a new master device. All databases—and all objects in every database—use the same logical page size.

See also

- *Configuration Parameters That Affect Memory Allocation* on page 36
- *Configuring Data Caches* on page 71
- *Explicitly Configuring the Default Cache* on page 76

How SAP ASE Allocates Memory

The size of SAP ASE logical pages (2, 4, 8, or 16K) determines the server's space allocation.

Each allocation page, object allocation map (OAM) page, data page, index page, text page, and so on are 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. 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 that have various sizes of logical pages within the same server. All tables are 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.

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 2K logical pages, `systypes`—with approximately 31 short rows, a clustered and a nonclustered index—reserves 3 extents, or 48K of memory. If you migrate the server to use 8K pages, the space reserved for `systypes` is still 3 extents, 192K of memory. For a server configured for 16K, `systypes` requires 384K 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. This table lists the minimum size for a database on each of the logical page sizes.

Table 2. Minimum database sizes

Logical page size	Minimum database size
2K	2MB
4K	4MB
8K	8MB
16K	16MB

Disk Space Allocation

The logical page size is not the same as the memory allocation page size. Memory allocation page size is always 2K, regardless of logical page size, which can be 2, 4, 8, or 16K.

Most memory-related configuration parameters use units of 2K for their memory page size, including:

- **max memory**
- **total logical memory**
- **total physical memory**
- **procedure cache size**
- **size of process object heap**
- **size of shared class heap**
- **size of global fixed heap**

How SAP ASE Allocates Buffer Pools

SAP ASE allocates buffer pools in units of logical pages.

For example, on a server using 2K logical pages, 8MB are allocated to the default data cache. This constitutes approximately 2048 buffers. If you allocated the same 8MB for the default data cache on a server using a 16K logical page size, the default data cache is approximately 256 buffers. On a busy system, this small number of buffers might result in a buffer always being in the wash region, causing a slowdown for tasks that request clean buffers. In general, to obtain the same buffer management characteristics on larger page sizes as with 2K logical page sizes, scale the size of the caches to the larger page size. So, if you increase your logical page size by four times, your cache and pool sizes should be about four times larger as well.

SAP ASE typically allocates memory dynamically and allocates memory for row processing as it needs it, allocating the maximum size for these buffers, even if large buffers are unnecessary. These memory management requests may cause SAP ASE to have a marginal loss in performance when handling wide-character data.

Heap Memory

A heap memory pool is an internal memory pool created at start-up that tasks use to dynamically allocate memory as needed. This memory pool is used by tasks that require large amounts of memory from the stack, such as tasks that use wide columns.

For example, if you make a wide column or row change, the temporary buffer this task uses can be as large as 16K, which is too big to allocate from the stack. SAP ASE dynamically allocates and frees memory during the task's runtime. The heap memory pool dramatically reduces the predeclared stack size for each task, while also improving the efficiency of memory usage in the server. The heap memory the task uses is returned to the heap memory pool when the task is finished.

Use the **heap memory per user** configuration parameter to set the heap memory.

Heap memory is measured in bytes per user. By default, the amount of memory is set to 4096 bytes. This example sets the value to 100 bytes per user:

```
sp_configure 'heap memory per user', 100
```

You can also specify the amount of memory in the number of bytes per user. For example, the following example specifies that each user connection is allocated 4K bytes of heap memory (the "0" is a placeholder **sp_configure** requires when you specify a unit value):

```
sp_configure 'heap memory per user', 0, "4K"
```

At the initial SAP ASE configuration, 1MB is set aside for heap memory. Additional heap memory is allocated for all the user connections and worker processes for which the server is configured, so the following configuration parameters affect the amount of heap memory available when the server starts:

- **number of user connections**
- **number of worker processes**

The global variable *@@heapmemsize* reports the size of the heap memory pool, in bytes.

Calculating Heap Memory

You can calculate how much heap memory SAP ASE sets aside (SAP ASE reserves a small amount of memory for internal structures, so these numbers vary from site to site):

Use this formula:

((1024 X 1024) + (*heap memory in bytes*)* (**number of user connections** + **number of worker processes**))

The initial value of (1024 X 1024) is the 1MB initial size of the heap memory pool. SAP ASE reserves a small amount of memory for internal structures.

For example, if your server is configured for:

- **heap memory per user** – 4K

- **number of user connections** – 25 (the default)
- **number of worker processes** – 25 (the default)

@@heapmemsize reports 1378304 bytes.

And the estimated value using the formula above, is:((1024 X 1024) + (4 X 1024 X 50)) = 1253376

Now, if you increase the **number of user connections**, the size of the heap memory pool increases accordingly:

```
sp_configure 'user connections', 100
```

@@heapmemsize reports 1716224 bytes.The estimated value in this case is:((1024 X 1024) + (4 * 1024 * (100 + 25)) = 1560576

If your applications fail with this message:

```
There is insufficient heap memory to allocate %ld bytes. Please
increase configuration parameter 'heap memory per user' or try again
when there is less activity on the system.
```

Increase the heap memory available to the server by increasing one of:

- **heap memory per user**
- **number of user connections**
- **number of worker processes**

The size of the memory pool depends on the number of user connections. SAP® recommends that you set heap memory per user to at least three times the size of your logical page.

SAP recommends that you first increase the **heap memory per user** configuration option before you increase **number of user connections** or **number of worker processes**. Increasing the **number of user connections** and **number of worker processes** consumes system memory for other resources, which may require you to increase the server's maximum memory.See *System Administration Guide: Volume 1 > Setting Configuration Paramters*.

Memory Management in SAP ASE

Memory exists in SAP ASE as total logical or physical memory.

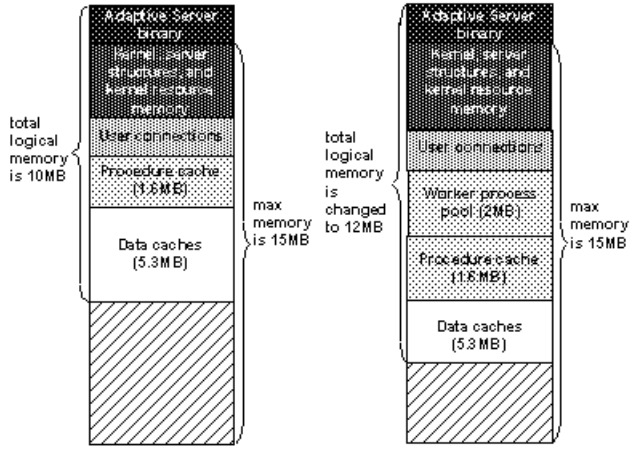
- Total logical memory – is the sum of the memory required for all the **sp_configure** parameters. The total logical memory must remain available, but may or may not be in use at a given moment. The total logical memory value may change due to changes in the configuration parameter values.
- Total physical memory – is the sum of all shared memory segments in SAP ASE. That is, total physical memory is the amount of memory SAP ASE uses at a given moment. You can verify this value with the read-only configuration parameter **total physical memory**. The value of **total physical memory** can only increase because SAP ASE does not shrink

memory pools once they are allocated. You can decrease the amount of total physical memory by changing the configuration parameters and restarting SAP ASE.

When SAP ASE starts, it allocates:

- Memory used by SAP ASE for nonconfigurable data structures
- Memory for all user-configurable parameters, including the data cache, the procedure cache, kernel resource memory, and the default data cache.

This figure illustrates how SAP ASE allocates memory as you change some of the memory configuration parameters:



When a 2MB worker process pool is added to the SAP ASE memory configuration, the procedure and data caches maintain their originally configured sizes; 1.6MB and 5.3MB, respectively. Because **max memory** is 5MB larger than the **total logical memory** size, it easily absorbs the added memory pool. If the new worker process pool brings the size of the server above the limit of **max memory**, any command you issue to increase the worker process pool fails. If this happens, the total logical memory required for the new configuration is indicated in the **sp_configure** failure message. Set the value of **max memory** to a value greater than the **total logical memory** required by the new configuration. Then retry your **sp_configure** request.

Note: The values for **max memory** and **total logical memory** do not include the SAP ASE binary.

The size of the default data cache and the procedure cache has a significant impact on overall performance. See *Performance and Tuning Series: Basics > Memory Use and Performance* for recommendations on optimizing procedure cache size.

Determining the Amount of Memory SAP ASE Needs

The total memory SAP ASE requires to start is the *sum of all memory configuration parameters* plus the *size of the procedure cache* plus the *size of the buffer cache,* where the size of the procedure cache and the size of the buffer cache are expressed in round numbers rather than in percentages.

The procedure cache size and buffer cache size do not depend on the total memory you configure. You can configure the procedure cache size and buffer cache size independently. Use **sp_cacheconfig** to obtain information such as the total size of each cache, the number of pools for each cache, the size of each pool, and so on.

Use **sp_configure** to determine the total amount of memory SAP ASE is using at a given moment:

```
1> sp_configure "total logical memory"
```

```
Parameter Name        Default   Memory Used   Config Value   Run Value
   Unit                Type
--------------        --------------- --- ----------- ------------
---------
--------------------- ----------
total logical memory  33792    127550         63775         63775
   memory pages(2k)   read-only
```

The value for the Memory Used column is represented in kilobytes, while the value for the Config Value column is represented in 2K pages.

The Config Value column indicates the total logical memory SAP ASE uses while it is running. The Run Value column shows the total logical memory being consumed by the current SAP ASE configuration. Your output differs when you run this command, because no two SAP ASEs are configured exactly the same.

See the *Reference Manual: Procedures.*

Determine the SAP ASE Memory Configuration

The total memory allocated during system start-up is the sum of memory required for all the configuration needs of SAP ASE. You can obtain this value from the read-only configuration parameter **total logical memory**.

This value is calculated by SAP ASE. The configuration parameter **max memory** must be greater than or equal to **total logical memory**. **max memory** indicates the amount of memory you will allow for SAP ASE needs.

During server start-up, by default, SAP ASE allocates memory based on the value of **total logical memory**. However, if the configuration parameter **allocate max shared memory** has been set, then the memory allocated will be based on the value of **max memory**. The configuration parameter **allocate max shared memory** enables a system administrator to

allocate the maximum memory that is allowed to be used by SAP ASE, during server start-up.

The key points for memory configuration are:

- The system administrator should determine the size of shared memory available to SAP ASE and set **max memory** to this value.
- The configuration parameter **allocate max shared memory** can be turned on during start-up and runtime to allocate all the shared memory up to **max memory** with the least number of shared memory segments. A large number of shared memory segments has the disadvantage of some performance degradation on certain platforms. Check your operating system documentation to determine the optimal number of shared memory segments. Once a shared memory segment is allocated, it cannot be released until the server is restarted.
- The difference between **max memory** and **total logical memory** determines the amount of memory available for the procedure and statement caches, data caches, or other configuration parameters.

  The amount of memory SAP ASE allocates during start-up is determined by either **total logical memory** or **max memory**. If you set **alloc max shared memory** to 1, SAP ASE uses the value for **max memory**.

  If either **total logical memory** or **max memory** is too high:
  - SAP ASE may not start if the physical resources on your machine are not sufficient.
  - If it does start, the operating system page fault rates may rise significantly and the operating system may need to be reconfigured to compensate.

If You Are Upgrading

In versions of SAP ASE earlier than 12.5, configuration values for **total logical memory**, **procedure cache percent,** and **min online engines** automatically calculate the new values for **procedure cache size** and **number of engines at startup**.

SAP ASE computes the size of the default data cache during the upgrade and writes this value to the configuration file. If the computed sizes of the data cache or procedure cache are smaller than the default sizes, they are reset to the default.

During the upgrade, SAP ASEs sets:

- **max memory** to the value of **total logical memory** specified in the configuration file. If necessary, reset the value of **max memory** to comply with the resource requirements.
- The number of engines in the previous configuration to the number of threads in `syb_default_pool`.

Use the **verify** option of **sp_configure** to verify any changes you make to the configuration file without having to restart SAP ASE:

```
sp_configure "configuration file", 0, "verify", "full_path_to_file"
```

Determining the Amount of Memory SAP ASE Can Use

There are differences between the upper limits of addressable shared memory for SAP ASE versions 12.0 and later versions.

Table 3. Addressable Memory Limits by Platform

Platform	32-bit SAP ASE	64-bit SAP ASE
HP-UX 11.x (PA-RISC processor)	2.75 gigabytes	16 EB[1]
IBM AIX 5.x	2.75 gigabytes	16 EB
Sun Solaris 8 (sparc processor)	3.78 gigabytes	16 EB
Sun Solaris 8 (Intel x86 processor)	3.75 gigabytes	N/A
Red Hat Enterprise Linux (Intel x86 processor)	2.7 gigabytes	N/A

[1]One exabyte (EB) equals 2^{60}, or 1024 petabyte. 16 exabyte is a theoretical limit; in practice, the actual value is limited by the total memory available on the system. SAP ASE has been tested with a maximum of 256GB of shared memory.

Starting Windows with the /3G option allows SAP ASE to use up to 3 gigabytes of shared memory. See your Windows documentation.

Note: SAP ASE 12.5 and later allocates memory differently than earlier versions. This includes changes to existing memory-related configuration parameters and introduces new memory parameters.

Each operating system has a default maximum shared-memory segment. Make sure the operating system is configured to allow the allocation of a shared-memory segment at least as large as **max memory**. See the *Installation Guide* for your platform.

Configuration Parameters That Affect Memory Allocation

When setting the SAP ASE memory configuration, use **sp_configure** to specify each memory requirement with an absolute value. Also specify the size of the procedure and default data caches with absolute values.

There are three configuration parameters that affect the way in which memory is allocated: **max memory**, **allocate shared memory**, and **dynamic allocation on demand**.

max memory

max memory allows you to establish a maximum setting for the amount of memory you can allocate to SAP ASE. Setting **max memory** to a slightly larger value than necessary provides extra memory that is utilized when SAP ASE memory needs increase.

When you increase the value for **max memory**, **sp_configure** sets **max memory** to the value you specify. However, memory allocation might happen later in time. The way SAP ASE allocates the memory specified by **max memory** depends on how you configure **allocate max shared memory** and **dynamic allocation on demand**.

When you upgrade, if the value for **max memory** is insufficient, SAP ASE automatically increases the value for **max memory**. The newer version of SAP ASE may require more memory because the size of internal data structures has increased.

allocate max shared memory

allocate max shared memory allows you to either allocate all the memory specified by **max memory** at start-up or to allocate only the memory required by the total logical memory specification during start-up.

On some platforms, if the number of shared memory segments allocated to an application is more than an optimal, platform-specific number, you may see some performance degradation. If this occurs, set **max memory** to the maximum amount available for SAP ASE. Set **allocate max shared memory** to 1 and restart the server. This ensures that all the memory for **max memory** is allocated by SAP ASE at start-up, using the smallest number of segments.

For example, if you set **allocate max shared memory** to 0 (the default) and **max memory** to 500MB, but the server configuration requires only 100MB of memory at start-up, SAP ASE allocates the remaining 400MB only when it requires the additional memory. However, if you set **allocate max shared memory** to 1, SAP ASE allocates the entire 500MB when it starts.

If **allocate max shared memory** is set to 0 and you increase **max memory**, the actual memory allocation happens when it is needed. If **allocate max shared memory** is set to 1and you increase **max memory**, SAP ASE attempts to allocate memory immediately. If the allocation fails, SAP ASE writes messages to the error log.

The advantage of allocating all memory at start-up is that there is no performance degradation while the server readjusts for additional memory. However, if you do not properly predict memory growth, and **max memory** is set to a large value, you may be wasting physical memory. Since you cannot dynamically decrease memory configuration parameters, it is important that you also consider other memory requirements.

dynamic allocation on demand

dynamic allocation on demand allows you to determine whether your memory resources are allocated as soon as they are requested, or only as they are needed. Setting **dynamic allocation on demand** to 1 allocates memory changes as needed, and setting it to 0 allocates the configured memory when you make changes to the memory configuration.

For example, if you set **dynamic allocation on demand** to 1, and change **number of user connections** to 1024, the **total logical memory** is 1024 multiplied by the amount of memory per user. If the amount of memory per user is 112K, the memory for user connections is 112MB (1024 x 112).

This is the maximum amount of memory that the **number of user connections** configuration parameter is allowed to use. However, if only 500 users are connected to the server, the amount of total physical memory used by the **number of user connections** parameter is 56MB (500 x 112).

If **dynamic allocation on demand** is 0, when you change **number of user connections** to 1024, all user connection resources are configured immediately.

Optimally, organize SAP ASE memory so that the amount of **total physical memory** is smaller than the amount of total logical memory, which is smaller than the **max memory**. This can be achieved, in part, by setting **dynamic allocation on demand** to 1, and setting **allocate max shared memory** to 0.

See also

- *Chapter 3, Configuring Memory* on page 29

Dynamically Allocating Memory

SAP ASE dynamically allocates physical memory, which means you can change the memory configuration without restarting the server.

Note: SAP ASE does not dynamically decrease memory. It is important that you accurately assess the needs of your system, because you may need to restart the server if you decrease the memory configuration parameters and want to release previously used physical memory.

Consider changing the value of the **max_memory** configuration parameter when:

- You change the amount of RAM on your machine.
- The pattern of use of your machine changes.
- The configuration fails because **max_memory** is insufficient.

If SAP ASE Cannot Start

When SAP ASE starts, it must acquire the full amount of memory set by **total logical memory**.

If SAP ASE cannot start because it cannot acquire enough memory, reduce the memory requirements by reducing the values for the configuration parameters that consume memory.

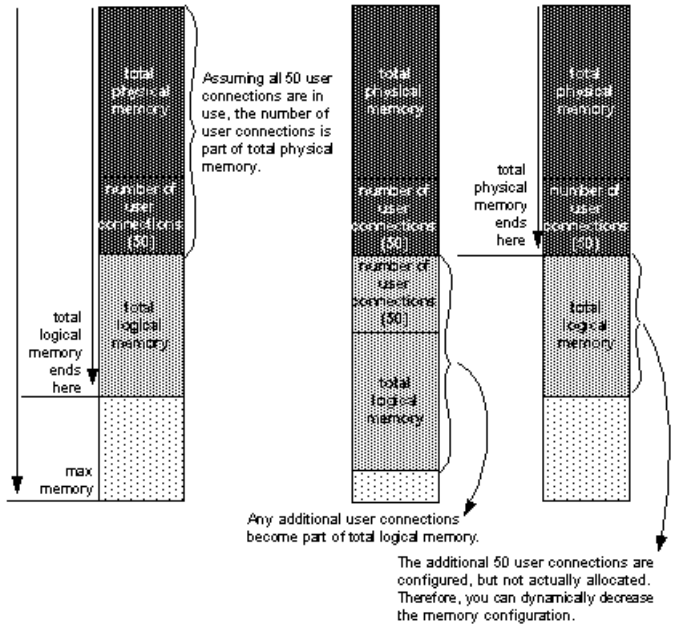
You may also need to reduce the values for other configuration parameters that require large amounts of memory. Restart SAP ASE to use the new values. See *System Administration Guide: Volume 1 > Setting Configuration Paramters*.

Dynamically Decreasing Memory Configuration Parameters

If you reset memory configuration parameters to a lower value, in-use memory is not dynamically released.

In this figure, because **dynamic allocation on demand** is set to 1, memory is now used only when there is an event that triggers a need for additional memory. In this example, such an event is a request for additional user connections, when a client attempts to log in to SAP ASE.

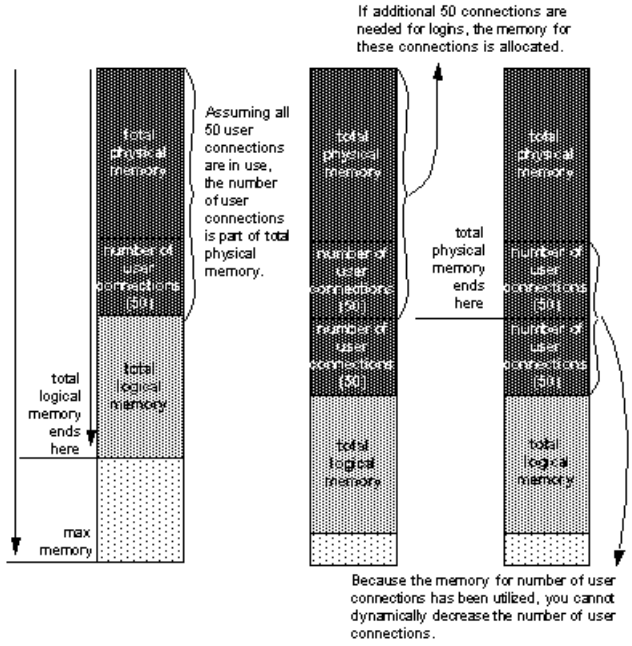
Figure 3: dynamic allocation on demand set to 1 with no new user connections



You may decrease **number of user connections** to a number that is greater than or equal to the number of user connections actually allocated, because, with **dynamic allocation on demand** set to 1, and without an actual increase in user connection request, no additional memory is required from the server.

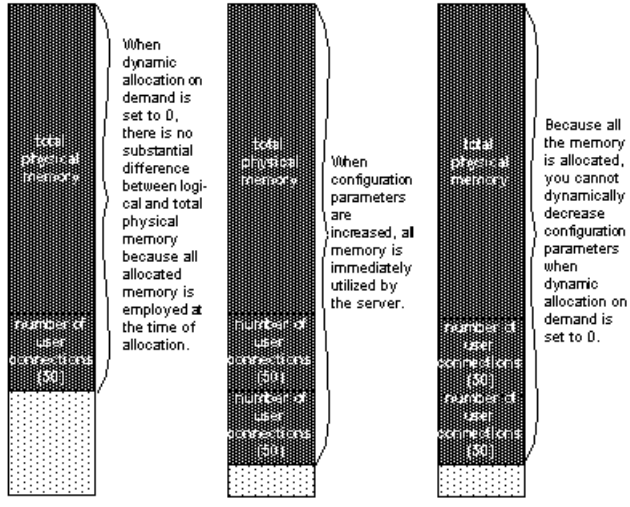
The figure below assumes that each of the additional 50 user connections is actually used. You cannot decrease **number of user connections** because the memory is in use. You can use **sp_configure** to specify a change to memory configuration parameters, but this change does not take place until you restart the server.

Figure 4: dynamic allocation on demand set to 1, with new user connections logged on



When **dynamic allocation on demand** is set to 0, all configured memory requirements are immediately allocated. You cannot dynamically decrease memory configuration.

Figure 5: dynamic allocation on demand set to 0



Note: In theory, when **dynamic allocation on demand** is set to 0, there should be no difference between total logical and physical memory. However, there are some discrepancies in the way that SAP ASE estimates memory needs, and the way in which memory is actually required for usage. For this reason, you may see a difference between the two during runtime.

In the previous examples, users can change the memory configuration parameter values to any smaller, valid value. This change does not take place dynamically, but it disallows new memory use. For example, if you have configured number of user connections to allow for 100 user connections and then change that value to 50 user connections (which occurs in the situations above), you can decrease the number of user connections value back to 50. This change does not affect the memory used by SAP ASE until after the server is restarted, but it prevents any new users from logging in to the server.

Configuring Thread Pools

SAP ASE records individual thread pool information in the configuration file under the `Thead Pool` heading.

By default, SAP ASE includes a set of system thread pools that are required for it to function. These include the `syb_default_pool` engine pool and multiple `run to completion` (RTC) thread pools, which do not contain engines. Users can create their own thread pools in addition to the system pools. User created thread pools are always engine pools.

At start-up, SAP ASE lists information for `syb_default_pool` and `syb_blocking_pool`, with their parameters set to the default values; the information for `syb_system_pool` is not included because Adpative Server calculates its number of threads at runtime, based on other configuration demands.

The configuration file lists these parameters for all thread pools:

- `number of threads` – configured number of threads in the pool.
- `description` – brief description of pool.

When SAP ASE starts, this is the default thread pool configuration:

```
[Thread Pool:syb_blocking_pool]
        number of threads = 4

[Thread Pool:syb_default_pool]
        number of threads = 1
```

As you add or remove thread pools, SAP ASE updates the configuration file, but does not require a restart for the changes to take effect. User-created thread pools (that you create with **create thread pool**) must be engine pools.

This example includes the two SAP-provided thread pools and a user-created thread pool, `sales_pool`:

```
[Thread Pool:sales_pool]
        description = pool for the sales force
        number of threads = 14
```

```
        idle timeout = 75

[Thread Pool:syb_blocking_pool]
         number of threads = 20

[Thread Pool:syb_default_pool]
        number of threads = 1
```

Use a file editor to edit the thread pool information in the configuration file, or use the **create thread pool**, **alter thread pool**, and **drop thread pool** commands to administer thread pools. See the *Reference Manual: Commands*.

If you edit the configuration file, SAP ASE starts using the new thread pool configuration, printing any change information to the log file (if you add thread pools with **create thread pool**, you need not restart SAP ASE). This output is from the log file after adding the `smaller_pool` thread pool to SAP ASE:

```
00:0000:00000:00000:2010/06/03 16:09:56.22 kernel  Create Thread
Pool 4, "smaller_pool", type="Engine (Multiplexed)", with 10 threads
```

In **isql**, use **create thread pool** to add thread pools. This example adds the `sales_pool` thread pool with 5 threads:

```
create thread pool sales_pool with thread count = 1
```

Use **sp_helpthread** to determine the runtime values for thread pools, including `syb_system_pool`. This is the **sp_helpthread** output for the thread pools above:

```
sp_helpthread
```

```
Name                Type                    Size   IdleTimeout
          Description
-----------------   --------------------  ----   -----------
          -----------------------------------------
sales_pool          Engine (Multiplexed)     1           100
          NULL
syb_blocking_pool   Run To Completion        4             0
          A pool dedicated to executing blocking calls
syb_default_pool    Engine (Multiplexed)     1           100
          The default pool to run query sessions
syb_system_pool     Run To Completion        4             0
          The I/O and system task pool
```

To drop the `sales_pool` thread pool, use:

```
drop thread pool sales_pool
```

See the *Reference Manual: Commands*.

You may see a message similar to this when you create a thread pool with an insufficient amount of memory (determined with **kernel resource memory**) to create the threads and there are an insufficient number of engines available:

```
00:0001:00000:00011:2010/06/11 14:46:38.32 kernel  Setting console
to nonblocking mode.
00:0001:00000:00011:2010/06/11 14:46:38.32 kernel  Create thread
pool pubs_pool
00:0001:00000:00011:2010/06/11 14:46:38.32 kernel  Create Thread
Pool 4, "pubs_pool", type="THREADPOOL_MULTIPLEXED", with 2 threads
00:0001:00000:00011:2010/06/11 14:46:38.32 kernel  could not
allocate memory for dynamic engine
00:0001:00000:00011:2010/06/11 14:46:38.32 kernel  Thread creation
failed to allocate an engine.
00:0001:00000:00011:2010/06/11 14:46:38.32 server  Configuration
file '/new_user/siena.cfg' has been written and the previous version
has been renamed to '/new_user/siena.009'.
1> 00:0025:00000:00000:2010/06/11 14:46:38.33 kernel  Network and
device connection limit is 1009.
00:0025:00000:00000:2010/06/11 14:46:38.33 kernel  ASE - Dynamic
Pluggable Component Interface is disabled
00:0025:00000:00000:2010/06/11 14:46:38.33 kernel  Encryption
provider initialization succeeded on engine 1.
00:0025:00000:00000:2010/06/11 14:46:38.33 kernel  Thread 25 (LWP
15726) of Threadpool pubs_pool online as engine 1
```

To create the thread pool, increase the value of **max online engines**, and restart SAP ASE.

See also

- *Configuring Engines in Threaded Mode* on page 110
- *Starting and Stopping Engines* on page 111

Determining the Total Number of Threads

The `monThread` monitor table includes information about all threads in SAP ASE.
To determine the total number of threads in SAP ASE, issue:

```
select count(*) from monThread
```

However, many threads reported from this query are from `syb_blocking_pool`, which do not require much CPU and need not be considered for resource planning.

Note: When you select entries for `monThread` (for example, counting the number of threads in the default thread pool), the number of threads SAP ASE reports may be fewer than the configured value because some threads may be changing states while SAP ASE collects the data. For example, a specific thread in the ACTIVE queue may place itself in the IDLE queue while the data collection is in progress.

Subsequent runs of the same query will provide exact results, unless some threads are transitioning their states.

During resource planning, pay particular attention to the number of query processing, or engine, threads, that run user queries. Determine the number of query processing threads by issuing `select count(*) from monEngine`, `select count(*) from`

`sysengines`, or by adding the thread counts for `syb_default_pool` to the thread counts for any user-created thread pools.

sp_sysmon displays the amount of CPU resources each thread consumes. See *Performance and Tuning Series: Monitoring SAP Adaptive Server with sp_sysmon*.

Tuning the syb_blocking_pool

Use the `monWorkQueue` monitoring table to determine an appropriate size for `syb_blocking_pool`.
Enlarge the size of `syb_blocking_pool` if it contains too many queued requests and requires a significant wait time.

Reduce the size of the pool if the `WaitCount` and `WaitTime` are zero.

sp_sysmon displays blocking pool statistics in the "Kernel" section of its output. You may need to increase the pool size if the "Blocking Call Activity" section shows a high percentage for "Queued Requests" relative to "Serviced Requests" or high "Total Wait Time."

System Procedures for Configuring Memory

Use the **sp_configure**, **sp_helpconfig**, and **sp_monitorconfig** system procedures to configure SAP ASE memory.

The full syntax and usage of **sp_configure** and details on each parameter are discussed in *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Viewing the Configuration Parameters for Memory

Issue the **Memory Use** configuration parameter to view the parameters associated with memory use on SAP ASE.

A "#" in the Memory Used column indicates that this parameter is a component of another parameter and that its memory use is included in the memory use for the other component. For example, memory used for **stack size** and **stack guard size** contributes to the memory requirements for each user connection and worker process, so the value is included in the memory required for **number of user connections** and for **number of worker processes**, if they are set to more than 200.

Some of the values from the Memory Use output are computed values. You cannot set them using **sp_configure**, but are reported to show where memory is allocated. Among the computed values is **total data cache size**.

Memory Available for Dynamic Growth

Issuing **sp_configure memory** displays all of the memory parameters and determines the difference between **max memory** and **total logical memory**, which is the amount of memory available for dynamic growth.

For example:

```
sp_configure memory
```

```
Msg 17411, Level 16, State 1:
Procedure 'sp_configure', Line 187:
Configuration option is not unique.
Parameter Name               Default  Memory Used  Config Value  Run
Value
Unit                         Type
------------------------------ ----------- ----------- ------------
----------
---------------------     ----------
additional network
memory                0           0             0           0
                bytes        dynamic
allocate max shared
memory                0           0             0           0
               switch        dynamic
compression memory
size                  0         152             0           0
     memory pages(2k)        dynamic
engine memory log size                    0
2            0          0
     memory pages(2k)        dynamic
heap memory per
user                  4096           0          4096       4096
                bytes        dynamic
kernel resource
memory                4096        8344          4096       4096
     memory pages(2k)        dynamic
lock shared
memory                        0           0             0           0
               switch         static
max
memory                       33792     300000      150000    150000
     memory pages(2k)        dynamic
memory alignment
boundary             16384           0        16384      16384
                bytes         static
memory per worker
process              1024           4          1024       1024
                bytes        dynamic
messaging
memory                       400           0          400        400
     memory pages(2k)        dynamic
pci memory
size                        32768           0        32768      32768
     memory pages(2k)        dynamic
shared memory starting
```

```
address            0           0            0          0
       not applicable          static
total logical
memory                   33792      110994        55497      55497
     memory pages(2k)       read-only
total physical
memory                       0       97656            0      48828
     memory pages(2k)       read-only
transfer utility memory
size           4096        8194         4096       4096
     memory pages(2k)         dynamic

An additional 30786 K bytes of memory is available for
reconfiguration. This is
the difference between 'max memory' and 'total logical memory'.
```

Using sp_helpconfig

sp_helpconfig estimates the amount of memory required for a given configuration parameter and value. It also provides a short description of the parameter, information about the minimum, maximum, and default values for the parameter, the run value, and the amount of memory used at the current run value.

You may find **sp_helpconfig** particularly useful if you are planning substantial changes to a server, such as loading large, existing databases from other servers, and you want to estimate how much memory is needed.

To see how much memory is required to configure a parameter, enter enough of the parameter name to make it unique, and the value you want to configure:

```
sp_helpconfig "worker processes", "50"
```

```
number of worker processes is the maximum number of worker processes
that can be
in use Server-wide at any one time.

Minimum Value  Maximum Value  Default Value  Current Value  Memory
Used
Unit     Type
-------------  -------------  -------------  -------------  --------
--
-------  ------------
0              2147483647     0              0              0
number   dynamic

Configuration parameter, 'number of worker processes', will consume
7091K of memory if configured at 50.
Changing the value of 'number of worker processes' to '50' increases
the amount of memory ASE uses by 7178 K.
```

You can also use **sp_helpconfig** to determine the value for **sp_configure**, if you know how much memory you want to allocate to a specific resource:

```
1> sp_helpconfig "user connections", "5M"
```

```
number of user connections sets the maximum number of user
connections that can
be connected to SQL Server at one time.
Minimum Value  Maximum Value  Default Value  Current Value  Memory
Used
Unit     Type
-------------  -------------  -------------  -------------  --------
--
-------  ------------
5              2147483647     25             25             3773
number  dynamic
Configuration parameter, 'number of user connections', can be
configured to 33
to fit in 5M of memory.
```

The important difference between the syntax of these two statements is the use of a unit of measure in the second example (the "M" for megabytes) to indicate to **sp_helpconfig** that the value is a size, not a configuration value. The valid units of measure are:

- **P** – pages (SAP ASE 2K pages)
- **K** – kilobytes
- **M** – megabytes
- **G** – gigabytes

There are some cases where the syntax does not make sense for the type of parameter, or where SAP ASE cannot calculate memory use. In these cases, **sp_helpconfig** prints an error message. For example, if you attempt to specify a size for a parameter that toggles, such as **allow resource limits**, **sp_helpconfig** prints the message that describes the function of the parameter for all the configuration parameters that do not use memory.

Using sp_monitorconfig

sp_monitorconfig displays metadata cache usage statistics for certain shared server resources.

These statistics include:

- The number of databases, objects, and indexes that can be open at any one time
- The number of auxiliary scan descriptors used by referential integrity queries
- The number of free and active descriptors
- The number of free memory pages
- The percentage of active descriptors
- The maximum number of descriptors used since the server was last started
- The current size of the procedure cache and the amount actually used
- The name of the instance on which you run **sp_monitorconfig**, if you are running in a clustered environment, or NULL if you are not running in a clustered environment.

For example, suppose the **number of open indexes** configuration parameter is 500. During a peak period, you can run **sp_monitorconfig** to get an accurate reading of the actual metadata cache usage for index descriptors:

```
sp_monitorconfig "number of open indexes"
```

```
Usage information at date and time: May 28 2010  1:12PM.
Name                        Num_free    Num_active  Pct_act  Max_Used
Reuse_cnt    Instance_Name
--------------------------  ----------  ----------  -------  ----------
----------  ------------------------------
number of open indexes           217         283    56.60         300
          0                                 NULL
```

The maximum number of open indexes used since the server was last started is 300, even though SAP ASE is configured for 500. Therefore, you can reset the **number of open indexes** configuration parameter to 330, which accommodates the 300 maximum used index descriptors, and allows for 10 percent more.

You can also determine the current size of the procedure cache using **sp_monitorconfig "procedure cache size"**. This parameter describes the amount of space the procedure cache is currently configured for and the most it has ever actually used. In this example, the procedure cache is configured for 20,000 pages:

```
sp_configure "procedure cache size"
```

```
Parameter Name                          DefaultMemory         Used
         Config Value          Run Value            Unit
         Type
------------------------------      -------------------   ----------
         -------------------   --------------------
---------------------
          --------------------- procedure cache
size                              7000        43914
                     20000               20000     memory pages(2k)
                       dynamic
```

However, when you run **sp_montorconfig "procedure cache size"**, you find that the maximum procedure cache ever used is 14241 pages, which means you can lower the run value of the procedure cache, saving memory:

```
sp_monitorconfig "procedure cache size"
```

```
Usage information at date and time: May 28 2010  1:35PM.
Name                        Num_free    Num_active  Pct_act  Max_Used
Reuse_cnt    Instance_Name
--------------------------  ----------  ----------  -------  ----------
----------  ------------------------------
procedure cache size          5878        14122    70.61       14241
         384                               NULL
```

View the number of free memory pages (`Num_free`) to determine if your kernel resource memory configuration is adequate:

```
sp_monitorconfig "kernel resource memory"
```

```
Usage information at date and time: Oct 12 2010  1:35PM.
Name                        Num_free    Num_active  Pct_act  Max_Used
Reuse_cnt    Instance_Name
--------------------------  ----------  ----------  -------  ----------
```

```
-----------  ------------------------------
kernel resource memory          9512         728     7.11       728
         0                                NULL
```

If the number of free pages is low, but the percent active (`Pct_act`) is high, you may need to increase the value of `kernel resource memory`.

Configuration Parameters That Control SAP ASE Memory

System administrators who are configuring SAP ASE for the first time should check configuration parameters that use large amounts of SAP ASE memory, and those that are commonly changed at a large number of installations.

Review these parameters when the system configuration changes, after upgrading to a new version of SAP ASE, or when making changes to other configuration variables that use memory.

SAP ASE Executable Code Size

The size of executable code is not included in the value of **total logical memory** or **max memory** calculations. **total logical memory** reports the actual memory requirement for SAP ASE configuration, excluding any memory that is required for the executable.

The memory requirements for executable code is managed by the operating system and is beyond the control of SAP ASE. Consult your operating system documentation.

Data and Procedure Caches

Specify the size of the data and procedure caches.

Determining the Procedure Cache Size

Use **procedure cache size** to specify, or view, the size of your procedure cache in 2K pages, regardless of the server's logical page size.
For example:

```
sp_configure "procedure cache size"
```

```
Parameter Name         Default  Memory Used  Config Value  Run Value
Unit              Type
--------------------  -------  -----------  ------------  ---------
--------------  ---------
procedure cache size     7000        15254          7000       7000
memory pages(2k)  dynamic
```

The amount of memory used for the procedure cache is appoximately 32.5KB (that is, 15254 2K pages plus overhead). To set the procedure cache to a different size, issue:

```
sp_configure "procedure cache size", new_size
```

This example resets the procedure cache size to 10000 2K pages (20MB):

```
sp_configure "procedure cache size", 10000
```

Determining the Default Data Cache Size

Both **sp_cacheconfig** and **sp_helpcache** display the current default data cache in megabytes. For example, the following shows an SAP ASE that is configured with 19.86MB of default data cache:

```
sp_cacheconfig
```

```
Cache Name              Status     Type       Config Value    Run Value
------------------       --------   --------   ------------    -----
-----
default data cache       Active     Default    0.00 Mb         19.86Mb
                                                ------------    --------
                      Total                    0.00Mb          19.86 Mb
================================================================
=====
Cache: default data cache, Status: Active, Type: Default
       Config Size: 0.00 Mb, Run Size: 19.86 Mb
       Config Replacement: strict LRU, Run Replacement: strict LRU
       Config Partition:    1,   Run Partition:    1
IO Size       Wash Size     Config Size     Run Size      APF Percent
--------      ---------     ------------    ----------    -----------
2 Kb           4066 Kb        0.00 Mb         19.86
Mb                 10
```

To change the default data cache, issue **sp_cacheconfig**, and specify "default data cache." For example, to change the default data cache to 25MB, enter:

```
sp_cacheconfig "default data cache", "25M"
```

This change is dymamic: you need not restart SAP ASE for the new value to take effect.

The default data cache size is an absolute value, and the minimum size for the cache size is 256 times the logical page size; for 2KB, the minimum is 512KB: for 16KB, the minimum is 4MB. The default value is 8MB. During an upgrade, SAP ASE sets the default data cache size to the value of the default data cache in the configuration file.

Monitoring Cache Space

Check data cache and procedure cache space with the **total data cache size** configuration parameter.

Another way to determine how SAP ASE uses memory is to examine the memory-related messages written to the error log when SAP ASE starts. These messages state exactly how much data and procedure cache is allocated, how many *compiled objects* can reside in cache at any one time, and buffer pool sizes.

These messages provide the most accurate information regarding cache allocation on SAP ASE. The amount of memory allocated for the procedure caches depends on the run value of the **procedure cache size** configuration parameter.

Each of these error log messages is described below.

- Procedure cache – this error log message provides information about the procedure cache:

  ```
  server: Number of blocks left for proc headers: 629
  ```

- Proc buffer – a proc buffer (procedure buffer) is a data structure that manages compiled objects in the procedure cache. One proc buffer is used for every copy of a compiled object stored in the procedure cache. When SAP ASE starts, it determines the number of proc buffers required and multiplies that value by the size of a single proc buffer (76 bytes) to obtain the total amount of memory required.
- Proc header – while in the procedure cache, a compiled object is stored in a proc header (procedure header). Depending on the size of the object to be stored, one or more proc headers may be required. The total number of compiled objects that can be stored in the procedure cache is limited by the number of available proc headers or proc buffers, whichever is less. The total size of procedure cache is the combined total of memory allocated to proc buffers (rounded up to the nearest page boundary), plus the memory allocated to proc headers.
- Data cache – when SAP ASE starts, it records the total size of each cache and the size of each pool in the cache in the error log. This example shows the default data cache with two pools and a user-defined cache with two pools:

  ```
  Memory allocated for the default data cache cache: 8030 Kb
  Size of the 2K memory pool: 7006 Kb
  Size of the 16K memory pool: 1024 Kb
  Memory allocated for the tuncache cache: 1024 Kb
  Size of the 2K memory pool: 512 Kb
  Size of the 16K memory pool: 512 Kb
  ```

Modify the ELC Size

The configuration option **engine local cache percent** configures Engine Local Cache (ELC) size for procedure cache memory.

Based on this configuration option, SAP ASE configures procedure cache ELC size as a percentage of **procedure cache size**.

- Default value for this configuration is 50. Which means ELC size will be 50% of **procedure cache size**.
- It is a static option; SAP ASE needs to be rebooted for the option to take effect.
- This configurations option can be used only when large ELC is enabled (using boot time traceflag 758).
- For optimal performance, a value no larger than 80 is recommended.

Example:

```
1> sp_configure "engine local cache percent",70
2> go
Parameter Name Default Memory Used Config Value Run Value Unit Type
-------------- ------- ----------- ------------ --------- ---- ----
    engine   local cache percent  50   0   70  50  percent static
(1 row affected)
```

Kernel Resource Memory

The **kernel resource memory** parameter determines the amount of memory available, in 2K pages, for internal kernel purposes, such as thread pools, tasks, and monitor counters.

The memory for **kernel resource memory** is drawn from **max memory**, which must be sufficiently large to accept a larger value for **kernel resource memory**.

User Connections

The amount of memory required per user connection varies by platform, and changes when you change some other configuration variables, including **default network packet size**, **stack size**, **stack guard size**, and **user log cache size**.

Changing any of these parameters changes the amount of space used by each user connection: multiply the difference in size by the number of user connections. For example, if you have 300 user connections, and you increase the **stack size** from 34K to 40K, the new value requires 1800K more memory.

Open Databases, Open Indexes, and Open Objects

The configuration parameters that control the total number of databases, indexes, partitions and objects that can be open simultaneously are managed by *metadata caches*, which reside in the server structures portion of memory.

Use these parameters to configure space for these caches:

- **number of open databases**
- **number of open indexes**
- **number of open objects**
- **number of open partitions**

See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

When SAP ASE opens a database or accesses an index, partition, or object, it reads information about it in the corresponding system tables: the information for databases is in `sysdatabases`, the information for indexes is in `sysindexes`, the information for partitions is in `syspartitions`, and so on.

The metadata caches for databases, indexes, partitions, or objects allow SAP ASE to access the information that describe them in `sysdatabases`, `sysindexes`, `syspartitions`, or `sysobjects` directly in its in-memory structure. In doing so, SAP ASE bypasses expensive calls that require disk access, thus improving performance. It also reduces synchronization and spinlock contention when SAP ASE needs to retrieve database, index, partition, or object information at runtime. Managing individual metadata caches is beneficial for a database that contains a large number of indexes, partitions, and objects, and where there is high concurrency among users.

Number of Locks

All processes in SAP ASE share a pool of lock structures.

As an initial estimate for configuring the number of locks, multiply the number of concurrent user connections you expect, plus the **number of worker processes** that you have configured, by 20.

The number of locks required by queries can vary widely. See *System Administration Guide: Volume 1 > Setting Configuration Parameters*.

SAP ASE issues error message 1204 if it runs out of locks, and only the "sa," or users with the sa_role, can log in to the server to configure additional locks. At this point, SAP ASE refuses login attempts by any other users and prints this message to the error log:

```
login: Process with spid <process id> could not connect to the ASE
which has temporarily run out of locks
```

While in this state, SAP ASE refuses login attempts from users for whom the sa_role is not automatically activated during login. Automatic role activation occurs if the system security office has taken any of these steps:

- Granted the sa_role directly to the user
- Granted the sa_role indirectly to the user through a user-defined role, which is specified as the user's default role
- Granted the sa_role to the user's login profile, and altered the profile to automatically activate the sa_role

Once logged in, the "sa" user, or the user with the sa_role, should immediately execute **sp_configure** to add locks. In this state, executing any other statement may cause SAP ASE to issue error message 1024 again.

SAP ASE may move to an unrecoverable state if a high number of "sa" users, or users with the sa_role, attempt to log in simultaneously when SAP ASE is out of locks.

Database Devices and Disk I/O Structures

The **number of devices** configuration parameter controls the number of database devices that can be used by SAP ASE for storing data.

When a user process must perform a physical I/O, the I/O is queued in a disk I/O structure.

See *System Administration Guide: Volume 1 > Setting Configuration Parameters*.

Parameters That Use Memory

SAP ASE includes a number of configuration parameters that use moderate amounts of memory.

Parallel Processing

Parallel processing requires more memory than serial processing.

The configuration parameters that affect parallel processing are:

- **number or worker processes** – sets the total number of worker processes available simultaneously in SAP ASE. Each worker process requires about the same amount of memory as a user connection.

  Changing any of these parameters changes the amount of memory required for each worker process:
  - default network packet size
  - **stack size** and **stack guard size**
  - **user log cache size**
  - **memory per worker process** – controls the additional memory that is placed in a pool for all worker processes. This additional memory stores miscellaneous data structure overhead and interworker process communication buffers.
- **memory per worker processes**
- **number of mailboxes** and **number of messages**

Each worker process makes its own copy of the query plan in space borrowed from the procedure cache. The coordinating process keeps two copies of the query plan in memory.

Remote Servers

Some configuration parameters that allow SAP ASE to communicate with other SAP servers such as Backup Server, Component Integration Services, or XP Server, use memory.

These parameters include **number of remote sites**, **number of remote sites**, **number of remote logins**, and **remote server pre-read packets**.

Set **number of remote sites** to the number of simultaneous sites you must communicate to or from on your server. If you use only Backup Server, and no other remote servers, you can increase your data cache and procedure cache space by reducing this parameter to 1. The connection from SAP ASE to XP Server uses one remote site.

These configuration parameters for remote communication use only a small amount of memory for each connection:

- **number of remote connections**
- **number of remote logins**

Each simultaneous connection from SAP ASE to XP Server for ESP execution uses one remote connection and one remote login.

Since the **remote server pre-read packets** parameter increases the space required for each connection configured by the **number of remote connections** parameter, increasing the number of pre-read packets can have a significant effect on memory use.

Referential Integrity

If the tables in your databases use a large number of referential constraints, you may need to adjust the **number of aux scan descriptors** parameter, if user connections exceed the default setting.

In most cases, the default setting is sufficient. If a user connection exceeds the current setting, SAP ASE returns an error message suggesting that you increase the **number of aux scan descriptors** parameter setting.

Parameters That Affect Memory

Other parameters affect memory. When you reset these configuration parameters, check the amount of memory they use and the effects of the change on your procedure and data cache.

- **additional network memory**
- **allow resource limits**
- **audit queue size**
- **event buffers per engine**
- **max number network listeners**
- **max online engines**
- **max SQL text monitored**
- **number of alarms**
- **number of large i/o buffers**
- **permission cache entries**

The Statement Cache

The statement cache saves SQL from cached statements.

SAP ASE compares incoming SQL statements to its cached SQL statements, and if they are equal, it executes the plan of the SQL statements already saved. This allows the application to amortize the costs of query compilation across several executions of the same statement.

Setting the Statement Cache

The statement cache lets SAP ASE compare a newly received ad hoc SQL statement to cached SQL statements. If a match is found, SAP ASE uses the plan cached from the initial execution.

In this way, SAP ASE does not have to recompile SQL statements for which it already has a plan.

The statement cache is a server-wide resource, which allocates and consumes memory from the procedure cache memory pool. Set the size of the statement cache dynamically using the **statement cache size** configuration parameter.

The maximum size of the statement cache depends on the version of SAP ASE:

- Versions 15.7 ESD #2 and later – allow you to store individual SQL statements up to 2MB (for a 64-bit machine) in the statement cache after you increase the **statement cache size** and **max memory** configuration parameters.
- Versions earlier than 15.7 ESD #2 – have a 16K limit on the size of individual statements stored in the statement cache, even if you configured **statement cache size** to a larger size.

Use **show_cached_text** to display the statement cache's SQL query text if it is less than 16K. However, if the SQL query text is larger than 16K, **show_cached_text** truncates the SQL query text, even if the full text is available in the statement cache.

Use **show_cached_text_long** to display SQL query text larger than 16K. **show_cached_text_long** displays SQL query text up to 2MB in size.

Note: If you deallocate or reduce the amount of memory for the statement cache, the original memory allocated is not released until you restart SAP ASE.

The syntax to set the size of the statement cache is as follows, where *size_of_cache* is the size, in 2K pages:

```
sp_configure "statement cache size", size_of_cache
```

For example, to set your statement cache to 5000 2K pages, enter:

```
sp_configure "statement cache size", 5000
```

See *System Administration Guide: Volume 1 > Setting Configuration Parameters*.

When you configure memory for the statement cache, consider:

- The amount of memory allocated for the procedure cache memory pool is the sum of the **statement cache size** and the **procedure cache size** configuration parameters. The statement cache memory is taken from the procedure cache memory pool.
- **statement cache size** limits the amount of procedure cache memory available for cached SQL text and plans. SAP ASE cannot use more memory for the statement cache than you have configured with the **statement cache size** configuration parameter.
- *@@nestlevel* contains the nesting level of current execution with the user session, initially 0. Each time a stored procedure or trigger calls another stored procedure or trigger, the nesting level is incremented. The nesting level is also incremented by one when a cached statement is created. If the maximum of 16 is exceeded, the transaction aborts.
- *@@procid* returns the ID of the currently executing stored procedure, and returns 0 for SQL statement. Once you enable the statement cache, the value fo *@@procid* is the value for the lightweight procedure.
- All procedure cache memory, including the memory allocated by the **statement cache size** configuration parameter, is available for stored procedures, which may replace cached statements on an least recently run (LRU) basis.
- Increase the **max memory** configuration parameter by the same amount configured for the statement cache. For example, if you have initially configured the statement cache size to 100 2K pages, increase **max memory** by the same amount.

- If you have used the **statement cache size** configuration parameter, you can disable and enable the statement cache at the session level with **set statement cache**. By default, the statement cache is on at the session level if it has been configured at the server level.
- Because each cached statement consumes one object descriptor, you must also increase the number of object descriptors accordingly, using the **number of open objects** configuration parameter.

Performing Ad Hoc Query Processing

When SAP ASE caches a statement, it changes the statement from an ad hoc query to a lightweight stored procedure.

For example, if you issue **sp_bindefault** or **sp_bindrule** in the same batch as the statement that invokes the default or rule without caching the statement, SAP ASE issues error message 540. However, if you cache the statement, SAP ASE binds the default or rule to the column.

SAP ASE may issue runtime errors instead of normalization errors when statements are cached and executed as stored procedures. For example, this query raises error number 241 if the statement is not cached, but raises a `Truncation error` and aborts the command if the statement is cached.

```
create table t1(c1 numeric(5,2)
go
insert t1 values(3.123)
go
```

To process ad hoc SQL statements using the statement cache:

1. SAP ASE parses the statement.

   If the statement should be cached, SAP ASE computes a hash value from the statement. SAP ASE uses this hash value to search for a matching statement in the statement cache.

   - If a match is found in the statement cache, SAP ASE skips to step 4.
   - If a match is not found, SAP ASE proceeds to step 2.
2. SAP ASE caches the SQL statement text.
3. SAP ASE wraps the SQL statement with a lightweight stored procedure and changes any local variables into procedure parameters. The internal representation of the lightweight procedure is not yet compiled into the plan.
4. SAP ASE converts the SQL statement into an **execute** statement for the corresponding lightweight procedure.
5. If there is no plan in the cache, SAP ASE compiles the procedure and caches the plan. SAP ASE compiles the plan using the assigned runtime values for the local variables. If the plan exists but is invalid, SAP ASE returns to step 3 using the text of the cached SQL statement
6. SAP ASE then executes the procedure. Substituting the lightweight procedure increments the *@@nestlevel* global variable.

Statement Matching Criteria

SAP ASE matches an ad hoc SQL statement to a cached statement by the SQL text and by login (particularly if both users have **sa_role**), user ID, database ID, and session state settings.

The relevant session state consists of settings for these **set** command parameters:

- **forceplan**
- **jtc**
- **parallel_degree**
- **prefetch**
- **quoted_identifier**
- **table count**
- **transaction isolation level**
- **chained** (transaction mode)

Settings for these parameters determine the behavior of the plan SAP ASE produces for a cached statement. See the *Reference Manual: Commands*.

Note: You must configure **set chained on/off** in its own batch if you enable the statement cache.

Caching Conditions

SAP ASE caches statements according to certain conditions.

These include:

- SAP ASE currently caches **select**, **update**, **delete**, and **insert select** statements with at least one table reference.
- The statement text cannot exceed 2MB in size.
- If a query includes an abstract plan (provided explicitly or through abstract plan **load**), and the statement cache is enabled and does not include an entry for the query, SAP ASE caches the plan obtained by the optimizer (and uses the abstract plan in case the abstract plan is legal).
- If a query includes an abstract plan (provided explicitly or through an abstract plan **load**) and the statement cache is enabled and includes an entry for the query, SAP ASE uses the plan corresponding to the query located in the statement cache, bypassing the optimizer and ignoring the abstract plan.
- If
  - The abstract plan capture feature in on (using **set plan dump on**)
  - The statement cache is enabled
  - A query comes which has an entry in the statement cache

  SAP ASE uses the corresponding plan in the statement cache, and bypasses the optimizer and the abstract plan capture feature. That is, SAP ASE performs no abstract plan capture and generates no abstract plan.

- SAP ASE does not cache **select into** statements, cursor statements, dynamic statements, plain **insert** (not **insert select**) statements, and statements within stored procedures, views, and triggers. Statements that refer to temporary tables are not cached, nor are statements with language parameters transmitted as binary large object (BLOB) datatypes. Statements that are prohibitively large are not cached. Also, **select** statements that are part of a conditional **if exists** or **if not exists** clause are not cached.

Statement Cache Sizing

Each cached statement requires approximately 1K memory in the statement cache, depending on the length of the SQL text.

Each cached plan requires at least 2K of memory in the procedure cache. To estimate the statement cache memory required, account for the following for each statement to be cached:

- The length of the SQL statement, in bytes, rounded up to the nearest multiple of 256.
- Approximately 100 bytes overhead.
- The size of the plan in the procedure cache. This size is equivalent to the size of a stored procedure plan containing only the cached statement. There may be duplicates of the plan for a single cached statement being used concurrently by two or more users.

Monitoring the Statement Cache

sp_sysmon reports on statement caching and stored procedure executions.

The statement cache is monitored by these counters:

- **Statements Cached** – the number of SQL statements added to the cache. This is typically the same number as **Statements Not Found**. Smaller values for **statements cached** means the statement cache is full of active statements.
- **Statements Found in Cache** – the number of times a query plan was reused. A low number of cache hits may indicate the statement cache is too small.
- **Statements Not Found** – indicates a lack of repeated SQL statements. The sum of **statements found in cache** and **statements not found** is the total number of eligible SQL statements submitted.
- **Statements Dropped** – the number of statements that were dropped from the cache. A high value may indicate an insufficient amount of procedure cache memory, or the statement cache size is too small.
- **Statements Restored** – the number of query plans regenerated from the SQL text. High values indicate an insufficient procedure cache size.
- **Statements Not Cached** – the number of statements SAP ASE would have cached if the statement cache were enabled. However, **Statements Not Cached** does not indicate how many unique SQL statements would be cached.

For example, this is sample output from **sp_sysmon**:

```
SQLStatement Cache:
   Statements
Cached                 0.0           0.0           0        n/a
   Statements Found in
```

```
Cache          0.7            0.0            2         n/a
   Statements Not
Found               0.0            0.0            0         n/a
   Statements
Dropped                 0.0            0.0            0         n/a
   Statements
Recompiled              0.3            0.0            1         n/a
   Statements Not
Cached              1.3            0.0            4        n/a
```

Aggregating Metrics from Syntactically Similar Queries

You can aggregate monitoring data for multiple statements in the statement cache. Although the statements are syntactically distinct—having individual comments or differences in their text format—represent semantically identical queries.

By providing a way to identify semantically equivalent statements, the **show_condensed_text** function can be used to aggregate the values from the `monCachedStatement` table into a single row in the query result.

`monCachedStatement` stores detailed monitoring information about the statement cache, including:

- Resources used during previous executions of a statement,
- How frequently a statement is executed
- The settings in effect for a particular plan
- The number of concurrent uses of a statement

This information can be helpful when you are troubleshooting, and when you are deciding which statements to retain in the cache. Although you can use **show_cached_text**, to view logically identical SQL statements that include nonunique SQL text, you may have difficulty identifying them in the output, especially if such statements are numerous. For example, if you have three queries that start with this text:

```
select * from db1.tableA where col1 in (?,?,?) additional_comment
select * from db1.tableA where col1 in (?,?,?)
different_additional_comment
select * from db1.tableA where col1 in (?,?,?)
different_additional_comment
```

Each query creates a separate entry in the global statement cache, compiles its own query plan, and produces three different SQL texts when you select from `monCachedStatement`.

However because these three queries have different SSQLIDs, metrics are distributed across multiple entries, and are reported separately in the top SQL statements monitor, resulting in:

- Expensive, but logically identical, SQL statements being unrealized, because they end up in multiple SSQLIDs, each of which shows only a part of the total elapsed time
- Logically identical SQL statements that use different query plans

Use **show_condensed_text** to group the statements reported in the `monCachedStatement` so the metrics for semantically identical statements can be aggregated together.

show_condensed_text uses these rules to condense SQL text (including SQL text larger than 16KB):

- Removes all comments and plan hints.
- Changes all keywords to upper case. However, object identifiers (for example, table names, column names, and so on) remain unchanged. For example, this statement:

  ```
  select id from sysobjects
  ```

  is condensed to:

  ```
  SELECT id FROM sysobjects
  ```
- Adds correlation names (using the format `T1`, `T2`, and so on) if the query contains multiple tables. For example, this statement:

  ```
  select error, id from sysmessages, sysobjects where id = error
  ```

  is condensed to:

  ```
  SELECT T1.error, T2.id FROM sysmessages T1, sysobjects T2 WHERE
  T2.id = T1.error
  ```
- Removes spaces:
  - Between tokens – keeps one space between tokens. All other spaces (for example, return characters, newline characters, tab characters, and so on) are removed. For example, the spaces in this statement are removed:

    ```
    select     name from sysdatabases
    ```

    and the statement is condensed to:

    ```
    select name from sysdatabases
    ```
  - From parenthesis – removes all spaces after the left parenthesis and before the right parenthesis. For example, this statement:

    ```
    select * from sysdatabases where name in ( ( select name from
    sysdevices ) )
    ```

    is condensed to:

    ```
    SELECT * FROM sysdatabases WHERE name IN ((SELECT name FROM
    sysdevices))
    ```
  - Before and after operators – keeps a single space before and after operators. For example, this statement:

    ```
    select error from sysmessages where error> 100
    ```

    is condensed to:

    ```
    SELECT error FROM sysmessages WHERE error > $
    ```
- Replaces all literals with $, regardless of their placement. For example, this statement:

  ```
  select name = ”mydb” from sysdatabases where name = ”master”
  ```

  is condensed to:

```
SELECT name = $ FROM sysdatabases WHERE name = $
```

- Reduces all **in**-lists to a single parameter. For example, this list:

  ```
  IN(100, 200, 300, 400)
  ```

  is condensed to:

  ```
  IN($)
  ```

  However, if the **in** list:
  - Does not contain a subquery – **show_condensed_text** reduces the **in**-list to a single parameter. For example, this query:

    ```
    select name from sysdatabases where name in(“master”, “tempdb”,
    “model”)
    ```

    is condensed to:

    ```
    SELECT name FROM sysdatabases WHERE name in ($)
    ```
  - Contains a subquery – **show_condensed_text** leaves the subquery, but other literals in the **in** lists are replaced with a $. For example, this query:

    ```
    select * from sysdatabases where name in (“master”, (select
    name from sysdatabases), “tempdb”)
    ```

    is condensed to:

    ```
    select * from sysdatabases where name in ($, (select name from
    sysdatabases), $)
    ```
- Changes **between** *value* **and** *value* to **>= AND <=**. For example, this statement:

  ```
  select name from sysdatabases where dbid between 1 and 5
  ```

  is condensed to:

  ```
  SELECT name FROM sysdatabases WHERE dbid >= $ AND dbid <= $
  ```
- Reduces repeated **UNION ALL** and **UNION** clauses to a single iteration. This example, which includes a series of UNION ALL clauses that are all the same:

  ```
  unique statement UNION ALL unique statement UNION ALL unique
  statement
  ```

  is condensed to:

  ```
  unique statement UNION ALL $
  ```

  However, **show_condensed_text** retains all **UNION ALL** clauses if they are distinct. For example, **show_condensed_text**keeps these **show_condensed_text** clauses:

  ```
   select name from sysdatabases
  union all select name from sysobjects
  union all select name from syscolumns
  ```

  But this statement:

  ```
   select name from sysdatabases
  where name = “master” union all select name
  from sysdatabases where name = “tempdb”
  union all select name from sysdatabases where name = “model”
  ```

  is condensed to:

```
SELECT name FROM sysdatabases WHERE name = $ UNION ALL $
```

- Reduces **or** clauses, even though they may have a different order than the original SQL text. For example, this statement:

```
 select count(*) from sysmessages where error<10 or error >
1000000 or error = 1000
```

  is condensed to:

```
SELECT COUNT(*) FROM sysmessages WHERE error = $ OR error < $ OR
error > $
```

- Applies the same rules for queries to subqueries.

Purging the Statement Cache

Run **dbcc purgesqlcache** to remove all the SQL statements from the statement cache. Any statements that are currently running are not removed.

You must have the **sa_role** to run **dbcc purgesqlcache**.

Printing Statement Summaries

Run **dbcc prsqlcache** to print summaries of the statements in the statement cache.

The *oid* option allows you to specify the object ID of the statement to print, and the *printopt* option allows you to specify whether you print the trace description (specify 0) or the **showplan** option (specify 1). If you do not include any values for *oid* or *printopt*, **dbcc prsqlcache** displays the entire contents of the statement cache.

You must have the **sa_role** to run **dbcc prsqlcache**.

This example specifies 1 in the **printopt** parameter for the **showplan** output:

```
dbcc prsqlcache (1232998109, 1)
```

This provides information for all statements in the cache:

```
Start of SSQL Hash Table at 0xfc67d830
Memory configured: 1000 2k pages            Memory used: 18 2k pages
Bucket# 625 address 0xfc67ebb8

SSQL_DESC 0xfc67f9c0
ssql_name *ss1248998166_0290284638ss*
ssql_hashkey 0x114d645e ssql_id 1248998166
ssql_suid 1             ssql_uid 1      ssql_dbid 1
ssql_status 0x28        ssql_parallel_deg 1
ssql_tab_count 0        ssql_isolate 1  ssql_tranmode 0
ssql_keep 0             ssql_usecnt 1   ssql_pgcount 8
SQL TEXT: select * from sysobjects where name like "sp%"

Bucket# 852 address 0xfc67f2d0
SSQL_DESC 0xfc67f840
ssql_name *ss1232998109_1393445479ss*
ssql_hashkey 0x530e4a67 ssql_id 1232998109
ssql_suid 1             ssql_uid 1      ssql_dbid 1
ssql_status 0x28        ssql_parallel_deg 1
ssql_tab_count 0        ssql_isolate 1  ssql_tranmode 0
```

```
ssql_keep 0             ssql_usecnt 1   ssql_pgcount 3
SQL TEXT: select name from systypes where allownulls = 0

End of SSQL Hash Table

DBCC execution completed. If DBCC printed error messages, contact a
user with
```

Or you can get information about a specific object ID:

```
dbcc prsqlcache (1232998109, 0)
```

```
SSQL_DESC 0xfc67f840
ssql_name *ss1232998109_1393445479ss*
ssql_hashkey 0x530e4a67 ssql_id 1232998109
ssql_suid 1             ssql_uid 1      ssql_dbid 1
ssql_status 0x28        ssql_parallel_deg 1
ssql_tab_count 0        ssql_isolate 1  ssql_tranmode 0
ssql_keep 0             ssql_usecnt 1   ssql_pgcount 3
SQL TEXT: select name from systypes where allownulls = 0

DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

```
SSQL_DESC 0xfc67f840
ssql_name *ss1232998109_1393445479ss*
ssql_hashkey 0x530e4a67 ssql_id 1232998109
ssql_suid 1             ssql_uid 1       ssql_dbid 1
ssql_status 0x28        ssql_parallel_deg 1
ssql_tab_count 0        ssql_isolate 1  ssql_tranmode 0
ssql_keep 0             ssql_usecnt 1   ssql_pgcount 3
SQL TEXT: select name from systypes where allownulls = 0

QUERY PLAN FOR STATEMENT 1 (at line 1).

      STEP 1
         The type of query is SELECT.
         FROM TABLE
            systypes
         Nested iteration.
         Table Scan.
         Forward scan.

         Positioning at start of table.
         Using I/O Size 2 Kbytes for data pages.
         With LRU Buffer Replacement Strategy for data pages.
         DBCC execution completed. If DBCC printed error messages,
         contact a user with
          System Administrator (SA) role.
```

Displaying the SQL Plan for Cached Statements

Use the **show_plan** function to view the plan for a cached statement.

The syntax is:

```
show_plan(spid, batch_id, context_id, statement_number)
```

Where:

- *spid* – process ID for any user connection.
- *batch_id* – unique number for a batch.
- *context_id* – unique number for every procedure (or trigger).
- *statement_number* – number of the current statement within a batch.

For a statement that is not performing well, you can change the plans by altering the optimizer settings or specifying an abstract plan.

When you specify the first `int` variable in the existing **show_plan** argument as "-1", **show_plan** treats the second parameter as a SSQLID.

Note: A single entry in the statement cache may be associated with multiple, and possibly different, SQL plans. **show_plan** displays only one of them.

CHAPTER 4 Configuring Data Caches

The most common reason for administering data caches is to reconfigure them for performance.

See the *Performance and Tuning Series: Basics > Memory Use and Performance* discusses performance concepts associated with data caches.

See also

- *Configuring Data Caches Using the Configuration File* on page 95

The SAP ASE Data Cache

The data cache holds the data, index, and log pages that are currently in use, as well as pages used recently by SAP ASE.

Immediately after installation, SAP ASE has a single default data cache that is used for all data, index, and log activity. The default size of this cache is 8M. Creating other caches does not reduce the size of the default data cache. Also, you can create pools within the named caches and the default cache to perform large I/Os. You can then bind a database, table (including the `syslogs` table), index, or text or image page chain to a named data cache.

Large I/O sizes enable SAP ASE to perform data prefetching when the query optimizer determines that prefetching would improve performance. For example, an I/O size of 128K on a server configured with 16K logical pages means that SAP ASE can read an entire extent—8 pages—all at once, rather than performing 8 separate I/Os.

Sorts can also take advantage of buffer pools configured for large I/O sizes.

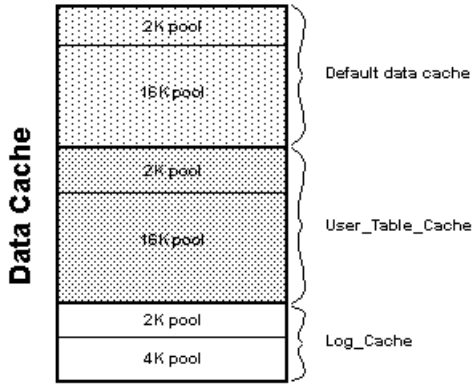
Configuring named data caches does not divide the default cache into separate cache structures. The named data caches that you create can be used only by databases or database objects that are explicitly bound to them. All objects not explicitly bound to named data caches use the default data cache.

SAP ASE provides user-configurable data caches to improve performance, especially for multiprocessor servers.

This figure shows a cache with the default data cache and two named data caches. This server uses 2K logical pages.

The default cache contains a 2K pool and a 16K pool. The `User_Table_Cache` cache also has a 2K pool and a 16K pool. The `Log_Cache` has a 2K pool and a 4K pool.

Figure 6: Data cache with default cache and two named data caches



Cache Configuration Commands and System Procedures

There are various system procedures and commands for configuring data caches.

This table lists commands and system procedures for configuring named data caches, for binding and unbinding objects to caches, and for reporting on cache bindings. It also lists procedures you can use to check the size of your database objects, and commands that control cache usage at the object, command, or session level.

Table 4. Commands and procedures for configuring named data caches

Command or procedure	Function
sp_cacheconfig	Creates or drops named caches, and changes the size, cache type, cache policy, or number of cache partitions.
sp_poolconfig	Creates and drops I/O pools, and changes their size, wash size, and asynchronous prefetch percent limit.
sp_bindcache	Binds databases or database objects to a cache.
sp_unbindc-ache	Unbinds specific objects or databases from a cache.
sp_unbindc-ache_all	Unbinds all objects bound to a specified cache.
sp_helpcache	Reports summary information about data caches, and lists the databases and database objects that are bound to caches.
sp_cachestrat-egy	Reports on cache strategies set for a table or index, and disables or reenables prefetching or MRU strategy.

Command or procedure	Function
sp_logiosize	Changes the default I/O size for the log.
sp_spaceused	Provides information about the size of tables and indexes, or the amount of space used in a database.
sp_estspace	Estimates the size of tables and indexes, given the number of rows the table will contain.
sp_help	Reports the cache to which a table is bound.
sp_helpindex	Reports the cache to which an index is bound.
sp_helpdb	Reports the cache to which a database is bound.
set showplan on	Reports on I/O size and cache utilization strategies for a query.
set statistics io on	Reports number of reads performed for a query.
set prefetch [on | off]	Enables or disables prefetching for an individual session.
select... (prefetch...lru | mru)	Forces the server to use the specified I/O size or MRU replacement strategy.

Most of the parameters for **sp_cacheconfig** that configure the data cache are dynamic and do not require that you restart the server for them to take effect.

In addition to using the commands to configure named data caches interactively, you can also edit the configuration file located in the `$SYBASE` directory. However, doing so requires that you restart the server.

See also

- *Configuring Data Caches Using the Configuration File* on page 95

Viewing Information About Data Caches

Use **sp_cacheconfig** to create and configure named data caches. When you first install SAP ASE, it has a single cache named `default data cache`.
To see information about caches, enter:

```
sp_cacheconfig
```

SAP ASE displays results similar to:

```
Cache Name                 Status    Type     Config Value  Run Value
-------------------------- --------- -------- ------------ ----------
--
default data cache         Active    Default       0.00 Mb    59.44 Mb
```

```
                                                 ------------ ------------
                                    Total          0.00 Mb    59.44 Mb
================================================================================
===
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 0.00 Mb,   Run Size: 59.44 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   12174 Kb     0.00 Mb     59.44 Mb      10
```

Summary information for each cache is printed in a block at the top of the report, ending with a total size for all configured caches. For each cache, there is a block of information reporting the configuration for the memory pools in the cache.

The columns are:

- Cache Name – gives the name of the cache.
- Status – indicates whether the cache is active. Possible values are:
  - "Pend/Act" – the cache was just created but not yet active.
  - "Active" – the cache is currently active.
  - "Pend/Del" – the cache is being deleted. The cache size was reset to 0 using **sp_cacheconfig** and **sp_poolconfig**.
- Type – indicates whether the cache can store data and log pages ("Mixed") or log pages only ("Log Only"). Only the default cache has the type "Default." You cannot change the type of the default data cache or change the type of any other cache to "Default."
- Config Value – displays the currently configured value. In the example output, the default data cache has not been explicitly configured, so its size is 0.
- Run Value – displays the size that SAP ASE is currently using.

The second block of output begins with three lines of information that describe the cache. The first two lines repeat information from the summary block at the top. The third line, "Config Replacement" and "Run Replacement" shows the cache policy, which is either "strict LRU" or "relaxed LRU." Run Replacement describes the setting in effect (either "strict LRU" or "relaxed LRU"). If the policy was changed since the server was restarted, the Config Replacement setting is different from the Run Replacement setting.

sp_cacheconfig then provides a row of information for each pool in the cache:

- IO Size – shows the size of the buffers in the pool. The default size of the pool is the size of the server's logical page. When you first configure a cache, all the space is assigned to the pool. Valid sizes are 2K, 4K, 8K, and 16K.
- Wash Size – indicates the wash size for the pool.
- Config Size and Run Size – display the configured size and the size currently in use. These may differ for other pools if you have tried to move space between them, and some of the space could not be freed.

- Config Partition and Run Partition – display the configured number of cache partitions and the number of partitions currently in use. These may differ if you have changed the number of partitions since last restart.
- APF Percent – displays the percentage of the pool that can hold unused buffers brought in by asynchronous prefetch.

A summary line prints the total size of the cache or caches displayed.

See also

- *Configuring Data Caches* on page 71
- *Changing the Wash Area for a Memory Pool* on page 86

Configuring Data Caches

Use an absolute value to specify the default data cache and the procedure cache for SAP ASE.

The first step in planning cache configuration and implementing caches is to set the **max memory** configuration parameter. After you set **max memory**, determine how much space to allocate for data caches on your server. The size of a data cache is limited only by access to memory on the system; however, **max memory** should be larger than **total logical memory**. You must specify an absolute value for the size of the default data cache and all other user-defined caches.

You can configure data caches in two ways:

- Interactively, using **sp_cacheconfig** and **sp_poolconfig**. This method is dynamic, which means you need not restart SAP ASE.
- By editing your configuration file. This method is static, which means you must restart SAP ASE.

Each time you change the data cache or execute either **sp_cacheconfig** or **sp_poolconfig**, SAP ASE writes the new cache or pool information into the configuration file and copies the old version of the file to a backup file. A message giving the backup file name is sent to the error log.

Some of the actions you perform with **sp_cacheconfig** are dynamic and some are static.

You can specify both static and dynamic parameters in a single command. SAP ASE performs the dynamic changes immediately and writes all changes to the configuration file (both static and dynamic). Static changes take effect the next time the server is started.

Table 5. Dynamic and static sp_cacheconfig actions

Dynamic sp_cacheconfig actions	Static sp_cacheconfig actions
Adding a new cache	Changing the number of cache partitions

Dynamic sp_cacheconfig actions	Static sp_cacheconfig actions
Adding memory to an existing cache	Reducing a cache size
Deleting a cache	Changing replacement policy
Changing a cache type	

You can reset static parameters by deleting and re-creating the cache:

1. Unbind the cache.
2. Delete the cache.
3. Re-create the cache using the new configuration.
4. Bind objects to the cache.

See also

- *Viewing Information About Data Caches* on page 69
- *Chapter 3, Configuring Memory* on page 29

Creating a New Cache

Use **sp_cacheconfig** to create a new cache.

See the *Reference Manual: Procedures*.

Maximum data cache size is limited only by the amount of memory available on your system. The amount of SAP ASE global memory dictates the memory required to create the new cache. When the cache is created:

- It has a default wash size.
- The asynchronous prefetch size is set to the value of **global async prefetch limit**.
- It has only the default buffer pool.

Use **sp_poolconfig** to reset these values.

To create a 10MB cache named `pubs_cache`:

```
sp_cacheconfig pubs_cache, "10M"
```

This command makes changes in the system tables and writes the new values to the configuration file. The cache is immediately active. Running **sp_cacheconfig** now yields:

```
sp_cacheconfigCache Name                    Status    Type     Config
Value Run Value
------------------------------ --------- -------- ------------
------------
default data cache           Active    Default      0.00 Mb     8.00
Mb
pubs_cache                   Active    Mixed       10.00 Mb    10.00
Mb
------------ ------------
```

```
Total      10.00 Mb     18.00 Mb
================================================================================
=======
Cache: default data cache,   Status: Active,   Type: Default
Config Size: 0.00 Mb,   Run Size: 8.00 Mb
Config Replacement: strict LRU,   Run Replacement: strict LRU
Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
4 Kb   1636 Kb      0.00 Mb      8.00 Mb     10
================================================================================
=======
Cache: pubs_cache,   Status: Active,   Type: Mixed
Config Size: 10.00 Mb,   Run Size: 10.00 Mb
Config Replacement: strict LRU,   Run Replacement: strict LRU
Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
4 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
```

The `pubs_cache` is now active, and all space is assigned to the smallest pool.

Note: When you create a new cache, the additional memory you specify is validated against **max memory**. If the sum of **total logical memory** and additional memory requested is greater than **max memory**, then SAP ASE issues an error and does not perform the changes.

You can create as many caches as you want without restarting SAP ASE.

Insufficient Space for New Cache

If SAP ASE cannot allocate all the memory requested, it allocates all the available memory and issues an error message.

```
ASE is unable to get all the memory requested (%d). (%d) kilobytes
have been allocated dynamically.
```

However, this additional memory is not allocated until the next restart of SAP ASE.

SAP ASE may notify you of insufficient space because some memory is unavailable due to resource constraints. System administrators should make sure these resource constraints are temporary. If the behavior persists, a subsequent restart may fail.

For example, if **max memory** is 700MB, `pub_cache` is 100MB, and the server's **total logical memory** is 600MB, and you attempt to add 100MB to `pub_cache`, the additional memory fits into **max memory**. However, if the server can allocate only 90MB, then it dynamically allocates this amount, but the `size field` of the cache in the configuration file is updated to 100MB. On a subsequent restart, since SAP ASE obtains memory for all data caches at one time, the size of `pub_cache` is 100MB.

See also

- *Adding Memory to an Existing Named Cache* on page 74

Adding Memory to an Existing Named Cache

Use **sp_cacheconfig** to add memory to an existing cache.

The additional memory you allocate is added to the SAP ASE page size pool. For example, in a server with a logical page size of 4K, the smallest size for a pool is a 4K buffer pool. If the cache has partitions, additional memory is divided equally among them.

If there is insufficient memory available, SAP ASE allocates what it can, then allocates the full amount when you restart the server.

For example, to add 2MB to a cache named `pub_cache` (the current size is 10MB), enter:

```
sp_cacheconfig pub_cache, "12M"
```

After you add the memory, **sp_cacheconfig** output is:

```
sp_cacheconfig pub_cache
```

```
 Cache Name                    Status    Type     Config Value Run Value
------------------------------ --------- -------- ------------
------------
pub_cache                     Active    Mixed     12.00 Mb      12.00
------------ ------------
Total      12.00 Mb     12.00 Mb
================================================================
=======
Cache: pub_cache,   Status: Active,   Type: Mixed
Config Size: 12.00 Mb,   Run Size: 12.00 Mb
Config Replacement: strict LRU,   Run Replacement: strict LRU
Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
4 Kb   2456 Kb      0.00 Mb     12.00 Mb     10
```

The change adds memory to the database page-size buffer pool and recalculates the wash size, as required. If the absolute value is set for wash size, SAP ASE does not recalculate it.

See also

- *Insufficient Space for New Cache* on page 73

Decreasing the Size of a Cache

When you reduce the size of a cache, you must restart SAP ASE for the change to take effect. This is a report on the `pubs_log` cache:

```
sp_cacheconfig pubs_log
```

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
pubs_log                 Active    Log Only      7.00 Mb      7.00 Mb
                                                ------------ ------------
                                     Total       7.00 Mb      7.00 Mb
```

```
============================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 7.00 Mb,   Run Size: 7.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    920 Kb      0.00 Mb      4.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

The following command reduces the size of the `pubs_log` cache to 6MB from a current size of 7MB:

```
sp_cacheconfig pubs_log, "6M"
```

After a restart of SAP ASE, **sp_cacheconfig** shows:

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
                                                ------------ ------------
                                     Total       6.00 Mb      6.00 Mb
============================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      3.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

When you reduce the size of a data cache, all the space to be removed must be available in the default pool (which is the smallest size available). You may need to move space to the default pool from other pools before you can reduce the size of the data cache. In the last example, to reduce the size of the cache to 3MB, use **sp_poolconfig** to move some memory into the default pool of 2K from the 4K pool. The memory is moved to "memory available for named caches."

See also

- *Changing the Size of Memory Pools* on page 90

Deleting a Cache

To completely remove a data cache, reset its size to 0.
For example:

```
sp_cacheconfig pubs_log, "0"
```

The cache is immediately deleted.

Note: You cannot drop the default data cache.

If you delete a data cache to which objects are bound, the cache is left in memory and SAP ASE issues:

```
Cache cache_name not deleted dynamically. Objects are bound to the
cache. Use sp_unbindcache_all to unbind all objects bound to the
cache.
```

The entry corresponding to the cache in the configuration file is deleted, as well as the entries corresponding to the cache in `sysconfigures`, and the cache is deleted the next time SAP ASE is restarted.

If you re-create the cache and restart SAP ASE, the bindings are marked valid again.

Use **sp_helpcache** to view all items bound to the cache. Use **sp_unbindcache_all** to unbind objects. See the *Reference Manual: Procedures*.

Explicitly Configuring the Default Cache

You must explicitly configure the size of the default data cache.

Use **sp_helpcache** to see the amount of memory remaining that can be used for the cache.

For example:

```
sp_helpcache
```

```
Cache Name                 Config Size    Run Size        Overhead
-----------------------   -------------  ----------     ----------
default data cache         50.00 Mb       50.00 Mb        3.65 Mb
pubs_cache                 10.00 Mb       10.00 Mb        0.77 Mb

Memory Available For       Memory Configured
Named Caches               To Named Caches
--------------------       ----------------
91.26 Mb                   91.25 Mb

------------------ Cache Binding Information: ------------------

Cache Name       Entity Name    Type          Index Name     Status
----------         -----------      -----           ---------
  ------
```

To specify the absolute size of the default data cache, execute **sp_cacheconfig** with default data cache and a size value. For example, to set the default data cache size to 25MB, enter:

```
sp_cacheconfig "default data cache", "25M"
```

When you re-run **sp_helpconfig**, “Config Value” shows the value.

```
sp_cacheconfig
```

```
Cache Name                 Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ----------
--
default data cache        Active    Default      25.00 Mb     50.00 Mb
```

```
pubs_cache                Active    Mixed        10.00 Mb     10.00 Mb
                                                ------------ ------------
                                      Total      10.00 Mb     60.00 Mb
================================================================================
===
Cache: default data cache,   Status: Active,   Type: Default
    Config Size: 25.00 Mb,   Run Size: 50.00 Mb
    Config Replacement: strict LRU,   Run Replacement: strict LRU
    Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb 10110 Kb     00.00 Mb    50.00 Mb      10
================================================================================
===
Cache: pubs_cache,   Status: Active,   Type: Mixed
    Config Size: 10.00 Mb,   Run Size: 10.00 Mb
    Config Replacement: strict LRU,   Run Replacement: strict LRU
    Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb    0.00 Mb    10.00 Mb    10
```

Use **sp_cacheconfig** to change the size of any named cache. Any changes you make to the size of any data cache do not affect the size of any other cache. Similarly, once you specify the size of the default data cache, the configuration of other user-defined caches does not alter the size of the default data cache.

Note: If you configure the default data cache and then reduce **max memory** to a level that sets the **total logical memory** value higher than the **max memory** value, SAP ASE does not start. Edit your configuration file to increase the size of other caches and increase the values of configuration parameters that require memory to create an environment in which **total logical memory** is higher than **max memory**.

Explicitly configure the default data cache and all user-defined caches with an absolute value. In addition, many configuration parameters use memory. To maximize performance and avoid errors, set the value of **max memory** to a level high enough to accommodate all caches and all configuration parameters that use memory.

SAP ASE issues a warning message if you set **max memory** to a value less than **total logical memory**.

See also

- *Chapter 3, Configuring Memory* on page 29

Changing the Cache Type

To reserve a cache for use by only the transaction log, change the cache's type to "logonly." The change is dynamic.

This example creates the cache `pubs_log` with the type "logonly:"

```
sp_cacheconfig pubs_log, "7M", "logonly"
```

This shows the initial state of the cache:

```
Cache Name           Status    Type    Config Value Run Value
-------------------- --------- -------- ------------ ------------
pubs_log             Pend/Act  Log Only      7.00 Mb      0.00 Mb
                                          ------------ ------------
                               Total              7.00 Mb      0.00 Mb
```

You can change the type of an existing "mixed" cache, as long as no non-log objects are bound to it:

```
sp_cacheconfig pubtune_cache, logonly
```

In high-transaction environments, SAP ASE usually performs best if the default value is twice the server's logical page size (for a server with 2K logical page size, it is 4K, for a server with 4K logical page size, it is 8K, and so on).. For larger page sizes (4, 8, and 16K) use **sp_sysmon** to find the optimum configuration for your site.

See also

- *Matching Log I/O Size for Log Caches* on page 82

Improving the Recovery Log Scan During load database and load tran

You can configure the default data cache to include a large buffer pool for reading log pages and asynchronous prefetch limits.

By default, boot time recovery automatically reconfigures the default data cache to include a large buffer pool for reading log pages and changes the asynchronous prefetch limits for both this pool and the default pool.

This is done to allow recovery to read log pages efficiently and to allow prefetch for large numbers of data (and index pages) and log pages. The settings applied by boot time recovery are:

- Large buffer pool (for example 128K for 16K page size) being 40% of the original page-sized pool size.
- The page size pool (for example 16K for a 16K page size) begin 60% of the original page-sized pool size.
- The **global async prefetch limit** for both pools set to 80% so that 80% of the pool can be used for pages that have been prefetched.

In SAP ASE 15.7 SP60 and later, specify whether or not to apply these same configuration changes during **load database** and **load tran** recovery. To apply the configuration changes, use the **sp_configure** configuration parameter **enable large pool for load**. The settings for **enable large pool for load** are:

- 0 - changes are not applied during **load database** and **load tran** recovery.

- 1 - changes are applied during **load database** and **load tran** recovery. When making this change, be aware that load recovery might then impact other live (production) databases that are using the default data cache.

Prior to 15.7 SP60, these changes were not applied to **load database** and **load tran** recovery because these commands are potentially done on a live system and the changes could negatively impact the rest of the system.

Configuring a Cache Replacement Policy

If a cache is dedicated to a table or an index, and has little or no buffer replacement when the system reaches a stable state, you can set the relaxed LRU (least recently used) replacement policy. This policy may improve performance for caches where there is little or no buffer replacement occurring, and for most log caches.

See *Performance and Tuning Series: Basics > Memory Use and Performance*.

To set relaxed replacement policy, use:

```
sp_cacheconfig pubs_log, relaxed
```

The default value is “strict.” The cache replacement policy and the asynchronous prefetch percentage are optional, but, if specified, they must have correct parameters or “DEFAULT”.

Note: Setting the cache replacement policy is static, requiring a restart of SAP ASE to take effect.

You can create a cache and specify its cache type and the replacement policy in one command. These examples create two caches, `pubs_log` and `pubs_cache`:

```
sp_cacheconfig pubs_log, "3M", logonly, relaxed
```

```
sp_cacheconfig pubs_cache, "10M", mixed, strict
```

Here are the results:

```
sp_cacheconfig
```

```
Cache Name                     Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
default data cache       Active    Default      25.00 Mb     42.29 Mb
pubs_cache               Active    Mixed        10.00 Mb     10.00 Mb
pubs_log                 Active    Log Only      7.00 Mb      7.00 Mb
                                                ------------ ------------
                                     Total      42.00 Mb     59.29 Mb
============================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
```

```
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb      0.00 Mb     42.29 Mb     10
================================================================
====
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb      0.00 Mb     10.00 Mb     10
================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 7.00 Mb,   Run Size: 7.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   1432 Kb      0.00 Mb      7.00 Mb          10
```

Dividing a Data Cache into Memory Pools

After you create a data cache, you can divide it into memory pools, each of which may have a different I/O size.

In any cache, you can have only one pool of each I/O size. The total size of the pools in any cache cannot be greater than the size of the cache. The minimum size of a memory pool is the size of the server's logical page. Memory pools larger than this must be a power of two and can be a maximum size of one extent.

When SAP ASE performs large I/Os, multiple pages ar simultaneously read into the cache. These pages are always treated as a unit; they age in the cache and are written to disk as a unit.

By default, when you create a named data cache, all of its space is assigned to the default memory pool. Creating additional pools reassigns some of that space to other pools, reducing the size of the default memory pool. For example, if you create a data cache with 50MB of space, all the space is assigned to the 2K pool. If you configure a 4K pool with 30MB of space in this cache, the 2K pool is reduced to 20MB.

After you create the pools, you can move space between them. For example, in a cache with a 20MB 2K pool and a 30MB 4K pool, you can configure a 16K pool, taking 10MB of space from the 4K pool.

The commands that move space between pools within a cache do not require you to restart SAP ASE, so you can reconfigure pools to meet changing application loads with little impact on server activity.

In addition to creating pools in the caches you configure, you can add memory pools for I/Os up to 16K to the default data cache.

The syntax for configuring memory pools is:

```
sp_poolconfig cache_name, "memsize[P|K|M|G]", "config_poolK" [,
"affected_poolK"]
```

The `config_pool` is set to the size specified in the command. The space is moved into or out of a second pool, the `affected_pool`. If you do not specify an `affected_pool`, the space is taken from or allocated to the 2K pool (the smallest size available). The minimum size for a pool is 512K.

This example creates a 7MB pool of 16K pages in the `pubs_cache` data cache:

```
sp_poolconfig pubs_cache, "7M", "16K"
```

To see the current configuration, run **sp_cacheconfig**, giving only the cache name:

```
sp_cacheconfig pubs_cache
```

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
pubs_cache               Active    Mixed        10.00 Mb     10.00 Mb
                                                ------------ ------------
                                     Total      10.00 Mb     10.00 Mb
=============================================================================
====
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 10.00 Mb,   Run Size: 10.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:             1,   Run Partition:             1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2048 Kb      0.00 Mb      3.00 Mb     10
   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
```

You can also create memory pools in the default data cache.

For example, start with this cache configuration:

```
Cache Name                  Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
default data cache       Active    Default      25.00 Mb     42.29 Mb
                                                ------------ ---
---------
                                      Total     25.00 Mb     42.29 Mb
=============================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:             1,   Run Partition:             1
```

```
IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb       0.00 Mb     42.29 Mb     10
```

If you then create a 16K pool in the default data cache that is 8MB:

```
sp_poolconfig "default data cache", "8M", "16K"
```

This is the resulting configuration, which has the “Run Size” of the 2K pool:

```
Cache Name                  Status    Type     Config Value Run Value
----------------------- --------- -------- ------------ -----------
-
default data cache       Active    Default      25.00 Mb     42.29 Mb
                                                ------------ ------------
                                     Total      25.00 Mb     42.29 Mb
================================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 42.29 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   8662 Kb       0.00 Mb     34.29 Mb     10
   16 Kb   1632 Kb       8.00 Mb      8.00 Mb     10
```

You need not configure the size of the 2K memory pool in caches that you create. Its Run Size represents all memory that is not explicitly configured to other pools in the cache.

Matching Log I/O Size for Log Caches

If you create a cache for the transaction log of a database, configure most of the space in that cache to match the log I/O size.

The default value is twice the server’s logical page size (for a server with 2K logical page size, it is 4K, for a server with 4K logical page size, it is 8K, and so on). SAP ASE uses 2K I/O for the log if a 4K pool is not available. Use **sp_logiosize** to change the log I/O size. The log I/O size of each database is reported in the error log when SAP ASE starts, or you can issue **sp_logiosize** with no parameters to check the size of a database.

This example creates a 4K pool in the **pubs_log** cache:

```
sp_poolconfig pubs_log, "3M", "4K"
```

You can also create a 4K memory pool in the default data cache for use by transaction logs of any databases that are not bound to another cache:

```
sp_poolconfig "default data cache", "2.5M", "4K"
```

See *Performance and Tuning Series: Basics > Choosing the I/O size for the transaction log*.

See also

- *Changing the Cache Type* on page 77

Binding Objects to Caches

Binding an object to a cache retains the object in memory and can expediate processing, keeping the bound objects in memory to avoid having any physical I/O when they are referenced.

sp_bindcache assigns a database, table, index, text object, or image object to a cache..Before you can bind an entity to a cache:

- The named cache must exist, and its status must be "Active."
- The database or database object must exist.
- To bind tables, indexes, or objects, you must be using the database where they are stored.
- To bind system tables, including the transaction log table `syslogs`, the database must be in single-user mode.
- To bind a database, you must be using the `master` database.
- To bind a database, user table, index, text object, or image object to a cache, the type of cache must be "Mixed." Only the `syslogs` table can be bound to a cache of "Log Only" type.
- You must own the object or be the database owner or the system administrator.

Binding objects to caches is dynamic; you need not restart the server.

The syntax for binding objects to caches is:

```
sp_bindcache cache_name, dbname [,[owner.]tablename
[, indexname | "text only" ] ]
```

This example binds the titles table to the `pubs_cache`:

```
sp_bindcache pubs_cache, pubs2, titles
```

To bind an index on `titles`, add the index name as the third parameter:

```
sp_bindcache pubs_cache, pubs2, titles, titleind
```

The owner name is not needed in the examples above because the objects in the `pubs2` database are owned by "dbo." To specify a table owned by any other user, add the owner name. You must enclose the entire parameter in quotation marks, since the period is a special character:

```
sp_bindcache pubs_cache, pubs2, "fred.sales_east"
```

This example binds the transaction log, `syslogs`, to the `pubs_log` cache:

```
sp_bindcache pubs_log, pubs2, syslogs
```

The database must be in single-user mode before you can bind any system tables, including the transaction log, `syslogs`, to a cache. Use **sp_dboption** from `master`, and a **use database** command, and run **checkpoint**:

```
sp_dboption pubs2, single, true
```

`text` and `image` columns for a table are stored in a separate data structure in the database. To bind this object to a cache, add the "text-only" parameter:

```
sp_bindcache pubs_cache, pubs2, au_pix, "text only"
```

This example, executed from `master`, binds the `tempdb` database to a cache:

```
sp_bindcache tempdb_cache, tempdb
```

You can rebind objects without dropping existing bindings.

Next

Binding caches includes restrictions. You cannot bind or unbind a database object when:

- Dirty reads are active on the object.
- A cursor is open on the object

In addition, SAP ASE must lock the object while the binding or unbinding takes place, so the procedure may have a slow response time, because it waits for locks to be released.

Getting Information About Cache Bindings

When you include the cache name, **sp_helpcache** provides information about a cache and the entities that are bound to it.
For example:

```
sp_helpcache pubs_cache
```

```
Cache Name           Config Size     Run Size       Overhead
-------------------- -------------   ----------     ----------
pubs_cache            10.00 Mb       10.00 Mb        0.77 Mb

------------------ Cache Binding Information: ------------------

Cache Name      Entity Name           Type   Index Name  Status
----------      -----------           ----   ----------  ------
pubs_cache      pubs2.dbo.titles      index  titleind    V
pubs_cache      pubs2.dbo.au_pix      index  tau_pix     V
pubs_cache      pubs2.dbo.titles      table              V
pubs_cache      pubs2.fred.sales_east table              V
```

If you use **sp_helpcache** without a cache name, it prints information about all the configured caches on SAP ASE and all the objects that are bound to them.

sp_helpcache performs string matching on the cache name, using `%cachename%`. For example, "pubs" matches both "`pubs_cache`" and "`pubs_log`".

The Status column reports whether a cache binding is valid ("V") or invalid ("I"). If a database or object is bound to a cache, and the cache is deleted, binding information is retained in the system tables, but the cache binding is marked as invalid. All objects with invalid bindings use the default data cache. If you subsequently create another cache with the same name, the

binding becomes valid when the cache is activated. The status of all user-defined caches must be “mixed cache” or “log only”.

Checking Cache Overhead

sp_helpcache can report the amount of overhead that is required to manage a named data cache of a given size.

When you create a named data cache, all the space you request with **sp_cacheconfig** is made available for cache space. The memory needed for cache management is taken from the global memory block pool.

Note: The cache overhead accounting for SAP ASE versions 12.5.1 and later is more explicit than in earlier versions. The amount of overhead is the same, but instead of being part of the server overhead, it is now considered, and reported, as part of the cache overhead.

To see the overhead required for a cache, include the proposed size. Use P for pages, K for kilobytes, M for megabytes, or G for gigabytes.

```
sp_helpcache "20000P"
```

This example checks the overhead for 20,000 pages:

```
2.96Mb of overhead memory will be needed to manage a cache of size
20000P
```

Configuring user caches does not waste cache space. Approximately 5 percent of memory is required for the structures that store and track pages in memory, whether you use a single large data cache or several smaller caches.

Effects of Overhead on Total Cache Space

Configuring a data cache can appear to increase or decrease the total available cache. This is due to the amount of overhead required to manage a cache of a particular size; the overhead is not included in the values that are shown by **sp_cacheconfig**.

The example in *Viewing Information About Data Caches*, above, shows a default data cache with 59.44 MB of available cache space before any user-defined caches are created. The server uses a 2K logical page. When the 10MB `pubs_cache` is created, the results of **sp_cacheconfig** show a total cache size of 59.44 MB.

Using **sp_helpcache** to check the overhead of the original 59.44MB default cache and the new 10MB cache shows that the change in space is due to changes in the size of overhead. The following example shows the overhead for the default data cache before any changes were made:

```
sp_helpcache "59.44M"
```

```
4.10Mb of overhead memory will be needed to manage a cache of size
59.44M
```

This example shows the overhead for `pubs_cache`:

```
sp_helpcache "10M"

0.73Mb of overhead memory will be needed to manage a cache of size
10M
```

This example shows the overhead for a cache size of 49.44MB:

```
sp_helpcache "49.44M"

3.46Mb of overhead memory will be needed to manage a cache of size
49.44M
```

4.19MB (which is equal to .73MB + 3.46MB) is the required overhead size for maintaining two caches of size 10MB and 49.44MB, and is slightly more than the 4.10MB overhead than is necessary to maintain one cache of 59.44MB.

Cache sizes are rounded to two places when printed by **sp_cacheconfig**, and overhead is rounded to two places by **sp_helpcache**, so the output includes a small rounding discrepancy.

Dropping Cache Bindings

When you drop a cache binding for an object, all pages that are currently in memory are cleared from the cache.

To drop cache bindings, use:

- **sp_unbindcache** to unbind a single entity from a cache.
- **sp_unbindcache_all** to unbind all objects bound to a cache.

The syntax for **sp_unbindcache** is:

```
sp_unbindcache dbname [,[owner.]tablename
[, indexname | "text only"] ]
```

This example unbinds the `titleidind` index on the `titles` table in the `pubs2` database:

```
sp_unbindcache pubs2, titles, titleidind
```

To unbind all the objects bound to a cache, use **sp_unbindcache_all**, giving the cache's name:

```
sp_unbindcache_all pubs_cache
```

Note: You cannot use **sp_unbindcache_all** if more than eight database objects in eight databases are bound to the cache. You must use **sp_unbindcache** on individual databases or objects to reduce the number of databases involved to eight or less.

Changing the Wash Area for a Memory Pool

A portion of each pool is configured as the *wash area*.

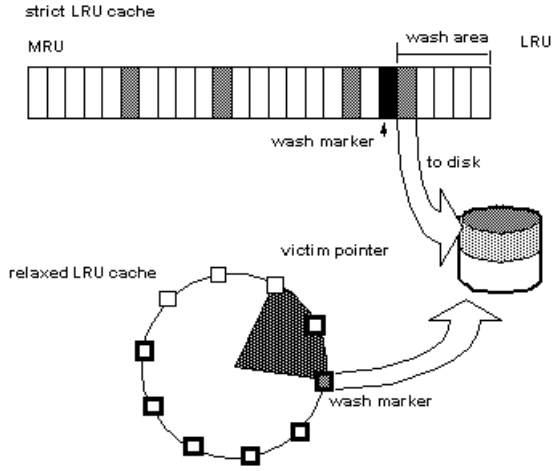
When SAP ASE must read a buffer into cache, it places the buffer either:

- At the LRU (least recently used) end of each memory pool, in a cache with strict LRU policy, or,
- At the victim pointer, in a cache with relaxed LRU policy. If the recently used bit of buffer at the victim marker is set, the victim pointer is moved to the next buffer in the pool.

After dirty pages (pages that have been changed in cache) pass the wash marker and enter the wash area, SAP ASE starts an asynchronous I/O on the page. When the write completes, the page is marked clean and remains available in the cache.

The space in the wash area must be large enough so that the I/O on the buffer can complete before the page needs to be replaced. This figure illustrates how the wash area of a buffer pool works with a strict and relaxed LRU cache.

Figure 7: Wash area of a buffer pool



By default, the size of the wash area for a memory pool is configured as follows:

- If the pool size is less than 300MB, the default wash size is 20 percent of the buffers in the pool.
- If the pool size is greater than 300MB, the default wash size is 20 percent of the number of buffers in 300MB.

The minimum wash size is 10 buffers. The maximum size of the wash area is 80 percent of the pool size. You must specify the pool size and wash size for all pools larger than 2KB

A buffer is a block of pages that matches the I/O size for the pool. Each buffer is treated as a unit: all pages in the buffer are read into cache, written to disk, and aged in the cache as a unit. For the size of the block, multiply the number of buffers by the pool size—for a 2KB pool, 256 buffers equals 512KB; for a 16KB pool, 256 buffers equals 4096KB.

For example, if you configure a 16K pool with 1MB of space, the pool has 64 buffers; 20 percent of 64 is 12.8. This is rounded down to 12 buffers, or 192K, is the size of the block that is allocated to the wash area.

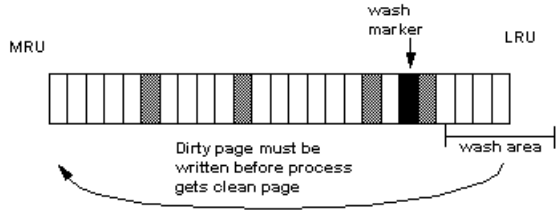
See also

- *Viewing Information About Data Caches* on page 69

When the Wash Area Is Too Small

If the wash area is too small for the usage in a buffer pool, operations that need a clean buffer may have to wait for I/O to complete on the dirty buffer at the LRU end of the pool or at the victim marker. This is called a *dirty buffer grab*, and it can seriously impact performance.

This figure shows a dirty buffer grab on a strict replacement policy cache.



Use **sp_sysmon** to determine whether dirty buffer grabs are taking place in your memory pools. Run **sp_sysmon** while the cache is experiencing a heavy period of I/O and heavy update activity, since it is the combination of many dirty pages and high cache replacement rates that usually causes dirty buffer grabs.

If the “Buffers Grabbed Dirty” output in the cache summary section shows a nonzero value in the “Count” column, check the “Grabbed Dirty” row for each pool to determine where the problem lies. Increase the size of the wash area for the affected pool.

This example sets the wash area of the 8K memory pool to 720K:

```
sp_poolconfig pubs_cache, "8K", "wash=720K"
```

If the pool is very small, you may also want to increase its size, especially if **sp_sysmon** output shows that the pool is experiencing high turnover rates.

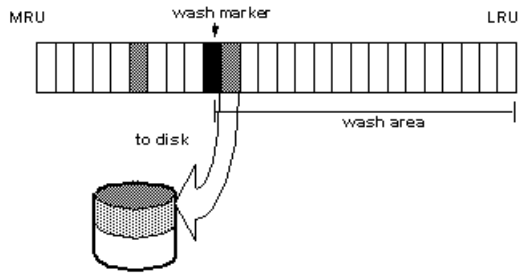
See the *Performance and Tuning Series: Monitoring SAP Adaptive Server with sp_sysmon*.

When the Wash Area is Too Large

If the wash area in a pool is too large, the buffers move too quickly past the wash marker in cache, and an asynchronous write is started on any dirty buffers.

This is described in this figure:

Figure 8: Effects of making the wash area too large



The buffer is marked clean and remains in the wash area of the MRU/LRU chain until it reaches the LRU. If another query changes a page in the buffer, SAP ASE must perform additional I/O to write the buffer to disk again.

If **sp_sysmon** output shows a high percentage of buffers "Found in Wash" for a strict replacement policy cache, and there are no problems with dirty buffer grabs, try reducing the size of the wash area. See the *Performance and Tuning Series: Monitoring SAP Adaptive Server with sp_sysmon*.

Setting the Housekeeper to Avoid Washes for Cache

You can use the **HK ignore cache** option of the **cache status** parameter to specify that you do not want the housekeeper to perform a wash on a particular cache.

Specifying caches to not include in the wash lets you avoid contention between the housekeeper and cache manager spinlock. Manually set **HK ignore cache** in the configuration file under each cache's heading; you cannot use **sp_cacheconfig** to set **HK ignore cache**.

This example from the configuration file shows the settings for the named cache `newcache`:

```
Named Cache:newcache
     cache size = 5M
     cache status = mixed cache
     cache status = HK ignore cache
     cache replacement policy = DEFAULT
local cache partition number = DEFAULT
```

You must set **HK ignore cache** along with either of **default data cache**, **mixed cache**, or **log parameters**. You cannot set the parameter **cache status** to only **HK ignore cache**.

Changing the Asynchronous Prefetch Limit for a Pool

The asynchronous prefetch limit specifies the percentage of the pool that can be used to hold pages that have been brought into the cache by asynchronous prefetch, but have not yet been used by any queries.

Use the **global async prefetch limit** parameter to set the default value for the server. Pool limits, which are set with **sp_poolconfig**, override the default limit for a single pool.

This command sets the percentage for the 2K pool in the `pubs_cache` to 20:

```
sp_poolconfig pubs_cache, "2K", "local async prefetch limit=20"
```

Changes to the prefetch limit for a pool take effect immediately and do not require a restart of SAP ASE. See *Performance and Tuning Series: Basics > Tuning Asynchronous Prefetch*.

Changing the Size of Memory Pools

Use **sp_poolconfig** to change the size of a memory pool, to specify the cache, the new size for the pool, the I/O size of the pool you want to change, and the I/O size of the pool from which the buffers should be taken.

If you do not specify the final parameter, all the space is taken from or assigned to the pool.

See also

- *Decreasing the Size of a Cache* on page 74

Moving Space from the Memory Pool

Use **sp_cacheconfig** to check the current configuration of the `pubs_log` cache.
For example, this output is based on the examples in the previous sections:

```
sp_cacheconfig pubs_log
```

```
Cache Name                Status    Type     Config Value Run Value
------------------------- --------- -------- ------------ ------------
-
pubs_log                  Active    Log Only      6.00 Mb      6.00 Mb
                                                ------------ ------------
                                    Total       6.00 Mb      6.00 Mb
================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
```

```
    2 Kb    716 Kb      0.00 Mb      3.50 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
```

To increase the size of the 4K pool to 5MB, moving the required space from the 2K pool, enter:

```
sp_poolconfig pubs_log, "5M", "4K"sp_cacheconfig pubs_log
```

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
-
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
                                                ------------ ------------
                                     Total       6.00 Mb      6.00 Mb
================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      1.00 Mb     10
    4 Kb   1024 Kb      5.00 Mb      5.00 Mb     10
```

Moving Space from Other Memory Pools

To transfer space from another pool specify the cache name, a “to” I/O size, and a “from” I/O size.

This output shows the current configuration of the default data cache:

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
-
default data cache       Active    Default      25.00 Mb     29.28 Mb
                                                ------------ ------------
                                     Total       25.00 Mb     29.28 Mb
================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      0.00 Mb     18.78 Mb     10
    4 Kb    512 Kb      2.50 Mb      2.50 Mb     10
   16 Kb   1632 Kb      8.00 Mb      8.00 Mb     10
```

To increase the size of the 4K pool from 2.5MB to 4MB, taking the space from the 16K pool:

```
sp_poolconfig "default data cache","4M", "4K","16K"
```

This results in:

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ -----------
-
default data cache       Active    Default      25.00 Mb     29.28 Mb
                                                ------------ ------------
                                     Total      25.00 Mb     29.28 Mb
================================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      0.00 Mb     18.78 Mb     10
    4 Kb    512 Kb      4.00 Mb      4.00 Mb     10
   16 Kb   1632 Kb      6.50 Mb      6.50 Mb     10
```

When you move buffers between pools in a cache, SAP ASE. It cannot move buffers that are in use or buffers that contain changes that have not been written to disk.

When SAP ASE cannot move as many buffers as you request, it displays an informational message, giving the requested size and the resulting size of the memory pool.

Adding Cache Partitions to Reduce Spinlock

On multiengine servers, more than one task can attempt to access the cache.

By default, each cache has a single spinlock, so that only one task can change or access the cache at a time. If cache spinlock contention is more than 10 percent, increasing the number of cache partitions for a cache may reduce spinlock contention, which improves performance.

To configure the number of cache partitions for:

- All data caches – use the **global cache partition number** configuration parameter
- An individual cache – use **sp_cacheconfig**

The number of partitions in a cache is always a power of 2 between 1 and 64. No pool in any cache partition can be smaller than 512K. In most cases, since caches can be sized to meet requirements for storing individual objects, you should use the local setting for the particular cache where spinlock contention is an issue.

See *Performance and Tuning Series: Basics > Reducing spinlock contention with cache partitions*.

Setting the Number of Cache Partitions

Use **sp_configure** to set the number of cache partitions for all caches on a server.

For example, to set the number of cache partitions to 2, enter:

```
sp_configure "global cache partition number",2
```

You must restart the server for the change to take effect.

Setting the Number of Local Cache Partitions

Use **sp_cacheconfig** or the configuration file to set the number of local cache partitions.

This command sets the number of cache partitions in the default data cache to 4:

```
sp_cacheconfig "default data cache", "cache_partition=4"
```

You must restart the server for the change to take effect.

Setting Precedence

The local cache partition setting always takes precedence over the global cache partition value.

These examples set the server-wide partition number to 4, and the number of partitions for `pubs_cache` to 2:

```
sp_configure "global cache partition number", 4
```

```
sp_cacheconfig "pubs_cache", "cache_partition=2"
```

The local cache partition number takes precedence over the global cache partition number, so `pubs_cache` uses 2 partitions. All other configured caches have 4 partitions.

To remove the local setting for `pubs_cache`, and use the global value instead:

```
sp_cacheconfig "pubs_cache", "cache_partition=default"
```

To reset the global cache partition number to the default, use:

```
sp_configure "global cache partition number", 0, "default"
```

Dropping a Memory Pool

To completely remove a pool, set its size to 0.
This example removes the 16K pool and places all space in the default pool:

```
sp_poolconfig "default data cache", "0", "16K"
```

```
Cache Name               Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
-
default data cache       Active    Default      25.00 Mb     29.28 Mb
                                                ------------ ------------
                                     Total      25.00 Mb     29.28 Mb
================================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 25.00 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
```

```
-------- --------- ------------ ------------ -----------
    2 Kb    3844 Kb       6.50 Mb      25.28 Mb      10
    4 Kb     512 Kb       4.00 Mb       4.00 Mb      10
```

If you do not specify the affected pool size (16K in the example above), all the space is placed in the default pool. You cannot delete the default pool in any cache.

When Pools Cannot Be Dropped Due to Page Use

If the pool you are trying to delete contains pages that are in use, or pages that have dirty reads, but are not written to disk, SAP ASE moves as many pages as possible to the specified pool and prints an informational message telling you the size of the remaining pool.

If the pool size is smaller than the minimum allowable pool size, you also receive a warning message saying the pool has been marked unavailable. If you run **sp_cacheconfig** after receiving one of these warnings, the pool detail section for these pools contains an extra Status column, that displays either “Unavailable/too small” or “Unavailable/deleted” in the Status column for the affected pool.

You can reissue the system procedure at a later time to complete removing the pool. Pools with “Unavailable/too small” or “Unavailable/deleted” are also removed when you restart SAP ASE.

Cache Binding Effects on Memory and Query Plans

Binding and unbinding objects may have an impact on performance.

When you bind or unbind a table or an index:

- The object’s pages are flushed from the cache – When you bind an object or database to a cache, the object’s pages that are already in memory are removed from the source cache. The next time the pages are needed by a query, they are read into the new cache. Similarly, when you unbind objects, the pages in cache are removed from the user-configured cache and read into the default cache the next time they are needed by a query.
- The object must be locked to perform the binding – To bind or unbind user tables, indexes, or text or image objects, the cache binding commands must have an exclusive table lock on the object. If a user holds locks on a table, and you issue **sp_bindcache**, **sp_unbindcache**, or **sp_unbindcache_all** on the object, the system procedure sleeps until it can acquire the locks it needs

  For databases, system tables, and indexes on system tables, the database must be in single-user mode, so there cannot be another user who holds a lock on the object.
- All query plans for procedures and triggers must be recompiled – Cache bindings and I/O sizes are part of the query plan for stored procedures and triggers. When you change the cache binding for an object, all the stored procedures that reference the object are recompiled the next time they are executed. When you change the cache binding for a

database, all stored procedures that reference any objects in the database that are not explicitly bound to a cache are recompiled the next time they are run

Configuring Data Caches Using the Configuration File

You can add or drop named data caches and reconfigure existing caches and their memory pools by editing the configuration file that is used when you start SAP ASE.

Note: You cannot reconfigure caches and pools on a server while it is running. Any attempt to read a configuration file that contains cache and pool configurations different from those already configured on the server causes the read to fail.

See also

- *Cache Configuration Commands and System Procedures* on page 68
- *Chapter 4, Configuring Data Caches* on page 67

Cache and Pool Entries in the Configuration File

Each configured data cache on the server has a block of information in the configuration file.

```
[Named Cache:cache_name]
     cache size = {size | DEFAULT}
     cache status = {mixed cache | log only | default data cache}
     cache replacement policy = {DEFAULT |
          relaxed LRU replacement| strict LRU replacement }
```

Size units can be specified with:

- P – pages (SAP ASE pages)
- K – kilobytes (default)
- M – megabytes
- G – gigabytes

This example shows the configuration file entry for the default data cache:

```
[Named Cache:default data cache]
      cache size = DEFAULT
      cache status = default data cache
      cache replacement policy = strict LRU replacement
```

The default data cache entry is the only cache entry that is required for SAP ASE to start. It must include the cache size and cache status, and the status must be “default data cache.”

If the cache has pools configured, the pool, the block in the preceding example is followed by a block of information for each pool:

```
[16K I/O Buffer Pool]
        pool size = size
        wash size = size
        local async prefetch limit = DEFAULT
```

Note: In some cases, there is no configuration file entry for the pool in a cache. If you change the asynchronous prefetch percentage with **sp_poolconfig**, the change is written only to system tables, and not to the configuration file.

This example shows output from **sp_cacheconfig**, followed by the configuration file entries that match this cache and pool configuration:

```
Cache Name                Status    Type     Config Value Run Value
------------------------ --------- -------- ------------ ------------
-
default data cache       Active    Default      29.28 Mb     25.00 Mb
pubs_cache               Active    Mixed        20.00 Mb     20.00 Mb
pubs_log                 Active    Log Only      6.00 Mb      6.00 Mb
tempdb_cache             Active    Mixed         4.00 Mb      4.00 Mb
                                                ------------ ------------
                                     Total      59.28 Mb     55.00 Mb
===========================================================================
====
Cache: default data cache,   Status: Active,   Type: Default
     Config Size: 29.28 Mb,   Run Size: 29.28 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   3844 Kb      6.50 Mb     25.28 Mb     10
    4 Kb    512 Kb      4.00 Mb      4.00 Mb     10
===========================================================================
====
Cache: pubs_cache,   Status: Active,   Type: Mixed
     Config Size: 20.00 Mb,   Run Size: 20.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb   2662 Kb      0.00 Mb     13.00 Mb     10
   16 Kb   1424 Kb      7.00 Mb      7.00 Mb     10
===========================================================================
====
Cache: pubs_log,   Status: Active,   Type: Log Only
     Config Size: 6.00 Mb,   Run Size: 6.00 Mb
     Config Replacement: relaxed LRU,   Run Replacement: relaxed LRU
     Config Partition:            1,   Run Partition:            1

IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    716 Kb      0.00 Mb      1.00 Mb     10
    4 Kb   1024 Kb      5.00 Mb      5.00 Mb     10
===========================================================================
====
Cache: tempdb_cache,   Status: Active,   Type: Mixed
     Config Size: 4.00 Mb,   Run Size: 4.00 Mb
     Config Replacement: strict LRU,   Run Replacement: strict LRU
     Config Partition:            1,   Run Partition:            1
```

```
IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
    2 Kb    818 Kb      0.00 Mb      4.00 Mb     10
```

This is the matching configuration file information:

```
[Named Cache:default data cache]
         cache size = 29.28M
         cache status = default data cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT

[2K I/O Buffer Pool]
        pool size = 6656.0000k
        wash size = 3844 K
        local async prefetch limit = DEFAULT

[4K I/O Buffer Pool]
        pool size = 4.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Named Cache:pubs_cache]
        cache size = 20M
        cache status = mixed cache
        cache replacement policy = strict LRU replacement
        local cache partition number = DEFAULT

[16K I/O Buffer Pool]
        pool size = 7.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Named Cache:pubs_log]
        cache size = 6M
        cache status = log only
        cache replacement policy = relaxed LRU replacement
        local cache partition number = DEFAULT

[4K I/O Buffer Pool]
        pool size = 5.0000M
        wash size = DEFAULT
        local async prefetch limit = DEFAULT
[Named Cache:tempdb_cache]
        cache size = 4M
        cache status = mixed cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT
```

Warning! Check the **max memory** configuration parameter and allow enough memory for other SAP ASE needs. If you assign too much memory to data caches in your configuration file, SAP ASE does not start. If this occurs, edit the configuration file to reduce the amount of space in the data caches, or increase the **max memory** allocated to SAP ASE.

See *System Administration Guide: Volume 1 > Setting Configuration Parameters*.

Cache Configuration Guidelines

There are a number of guidelines to follow when configuring user-definable caches.

- Make sure that your default data cache is large enough for all cache activity on unbound tables and indexes. All objects that are not explicitly bound to a cache use the default cache. This includes any unbound system tables in the user databases, the system tables in `master`, and any other objects that are not explicitly bound to a cache.
- During recovery, only the default cache is active. All transactions that must be rolled back or rolled forward must read data pages into the default data cache. If the default data cache is too small, it can slow recovery time.
- Do not "starve" the 2K pool in any cache. For many types of data access, there is no need for large I/O. For example, a simple query that uses an index to return a single row to the user might use only 4 or 5 2K I/Os, and gain nothing from 16K I/O.
- Certain **dbcc** commands and **drop table** can perform only 2K I/O. **dbcc checktable** can perform large I/O, and **dbcc checkdb** performs large I/O on tables and 2K I/O on indexes.
- For caches used by transaction logs, configure an I/O pool that matches the default log I/O size. This size is set for a database using **sp_logiosize**. The default value is 4K.
- Trying to manage every index and object and its caching can waste cache space. If you have created caches or pools that are not optimally used by the tables or indexes bound to them, they are wasting space and creating additional I/O in other caches.
- If `tempdb` is used heavily by your applications, bind it to its own cache. You can bind only the entire `tempdb` database—you cannot bind individual objects from `tempdb`.
- For caches with high update and replacement rates, be sure that your wash size is large enough.
- On multi-CPU systems, spread your busiest tables and their indexes across multiple caches to avoid spinlock contention.
- Consider reconfiguring caches or the memory pools within caches to match changing workloads. Reconfiguring caches requires a restart of the server, but memory pool reconfiguration does not.

  For example, if your system performs mostly OLTP (online transaction processing) during most of the month, and has heavy DSS (decision-support system) activity for a few days, consider moving space from the 2K pool to the 16K pool for the high DSS activity and resizing the pools for OLTP when the DSS workload ends.

Configuration File Errors

If you edit your configuration file manually, carefuly check the cache, pool, and wash sizes.

The total size of all of the caches cannot be greater than the amount of **max memory**, minus other SAP ASE memory needs.

In most cases, problems with missing entries are reported as "unknown format" errors on lines immediately following the entry where the size, status, or other information has been omitted.

Other errors provide the name of the cache in which the error occurred and the type of error. For example, you see this error if the wash size for a pool is specified incorrectly:

```
The wash size for the 4k buffer pool in cache pubs_cache has been
incorrectly configured. It must be a minimum of 10 buffers and a
maximum of 80 percent of the number of buffers in the pool.
```

CHAPTER 5 Managing Multiprocessor Servers

SAP ASE uses the Virtual Server Architecture™, which enables it to take advantage of the parallel processing feature of symmetric multiprocessing (SMP) systems.

You can run SAP ASE as a single process, as a single, multithreaded process, or as multiple, cooperating processes, depending on the number of CPUs available and the demands placed on the server machine. This chapter describes:

- The target machine architecture for the SMP SAP ASE
- SAP ASE architecture for SMP environments
- SAP ASE task management in the SMP environment
- Managing multiple engines

For information on application design for SMP systems, see *Performance and Tuning Series: Basics > Using Engines and CPUs*.

SAP ASE Kernels

SAP ASE version 15.7 and later includes a threaded kernel and a process kernel.

The kernel for which you configure SAP ASE determines the mode in which SAP ASE runs:

- Threaded mode – SAP ASE runs as a single multithreaded operating system process, and processes SQL queries with engines running on threads in thread pools. Threaded mode utilizes threads without engines to manage I/O. Administrators can configure additional thread pools to manage workload.
- Process mode – SAP ASE runs as multiple operating system processes that cooperate to work as a single server. Process mode uses engines to manage I/O, and administrators configure engine groups to manage workload.

Note: Process mode is not available on Windows.

For many workloads, threaded mode uses significantly less CPU than process mode, delivering the same—or better—performance. Threaded mode does not require as much task-to-engine affinity, thereby delivering more consistent performance in a mix of I/O- and CPU-intensive workloads.

The threaded kernel allows SAP ASE to take advantage of parallel hardware and support systems that have more processors, processor cores, and hardware threads than earlier-version kernels.

Although version 15.7 changes the kernel, the query processor remains the same. To run in threaded kernel mode, you need not change most scripts written for earlier versions of SAP

ASE, although few commands and stored procedures have changed. Applications are completely compatible with threaded mode.

Target Architecture

The SMP product is intended for machines with a specific set of features, including a symmetric multiprocessing operating system, shared memory over a common bus, 1–1024 processors, cores, or hardware threads, no master processor, and very high throughput.

SAP ASE consists of one or more cooperating processes that are scheduled onto physical CPUs by the operating system.

SAP ASE uses multiple cooperative processes to leverage parallel hardware when running in process mode, and uses multiple threads from the same process when running in threaded mode. Each process in the process-mode kernel is an SAP ASE engine. The threaded-mode kernel uses some threads as engines, and has additional nonengine threads.

Process mode uses multithreaded processes. However, because SAP ASE performs most of its work in the main thread of each process, consider these processes to be single-threaded when researching and tuning CPU resources.

SAP ASE uses engines as processors to execute SQL queries. In process mode, the engine is the main thread for each process. In threaded mode, the engines are threads from one or more engine thread pools. Multiple engines use shared memory to communicate. Process and threaded mode both use shared memory, including single engine or uniprocessor environments.

The operating system schedules SAP ASE threads onto CPU resources. SAP ASE does not distinguish between physical processors, cores, or subcore threads.

When configured for kernel mode, SAP ASE executes the engines within the thread of the single operating system process. SAP ASE acquires threads used to support engines from engine thread pools.

SAP ASE includes other nonengine threads that are used for particular tasks, but are not considered to be engines.

Figure 9: Threaded-Mode Architecture

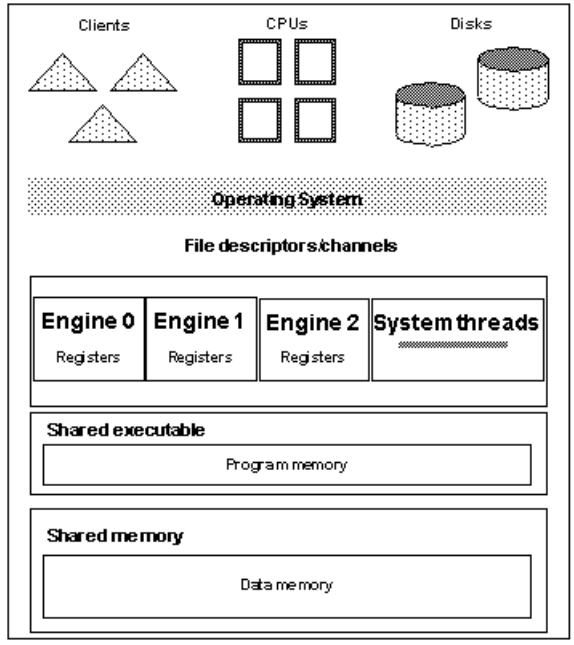
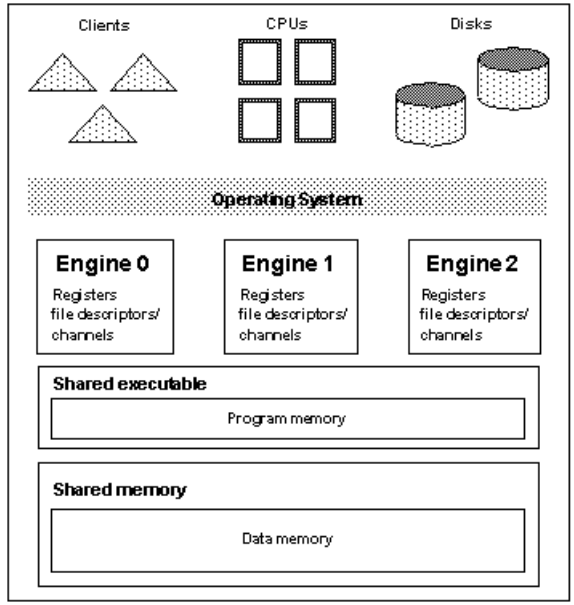


Figure 10: Process-Mode Architecture



The operating system schedules SAP ASE threads (engine and non-engine) onto physical CPU resources, which can be processors, cores, or subcore threads. The operating system—not SAP ASE—assigns threads to CPU resources: SAP ASE performance depends on receiving CPU time from the operating system.

SAP ASE engines perform all database functions, including updates and logging. SAP ASE—not the operating system—dynamically schedules client tasks to available engines. Tasks are execution environments within SAP ASE.

"Affinity" is a process in which certain SAP ASE tasks run only on a certain engine (task affinity), certain engines handle network I/O for a certain task (network I/O affinity), or certain engines run only on a certain CPU (engine affinity).

In process mode, a connection to SAP ASE has network I/O affinity with the engine that accepted the connection. This engine must do all the network I/O for that connection. Network I/O affinity does not exist in threaded mode because any engine can perform the I/O for any connection, which typically reduces context switching and improves performance.

You can use the logical process manager to establish task affinity so the manager runs a task, or set of tasks, only on a specific engine or specific set of engines. In threaded mode, use thread pools to accomplish task affinity. In process mode, use engine groups.

Thread pools and engine groups have different behaviors. Engines in a thread pool execute tasks assigned only to that thread pool. The scheduler search space (the area in which the scheduler searches for runnable tasks) is limited to engines in that thread pool. Engines in an engine group may run any task, as long as the task is not restricted to engines in a different group. That is, including a task in an engine group restricts where the task may run, but does not reserve the engines in the group for that task.

Configure task affinity using the SAP ASE logical process manager (see *Performance and Tuning Series: Basics > Distributing Engine Resources*). Configure engine or CPU affinity with **dbcc tune** or equivalent operating system commands (see the *Reference Manual: Commands* and your operating system documentation).

Kernel Modes

By default, SAP ASE starts in threaded mode, appearing as a single process on your operating system, and using native operating system threads for parallelism. By default, SAP ASE handles I/O with native threads.

In process mode, SAP ASE appears as multiple processes on the operating system.

Process mode may not support features available in later versions of SAP ASE. SAP intends for you to use process mode as a fallback if you experience problems running in threaded mode.

Note: For architecture reasons, process mode is not available on the Windows platform. Earlier versions of SAP ASE on Windows also used a multithreaded, single-process kernel.

To determine the current mode of SAP ASE, use:

```
select @@kernelmode

-------------------------------------
threaded
```

Switching Kernel Modes

Use **sp_configure "kernel mode"** to switch the kernel mode.
This example switches SAP ASE to the process mode:

```
sp_configure "kernel mode", 0, process
```

When you switch modes, SAP ASE issues a message similar to:

```
Parameter Name                  Default              Memory Used
        Config Value         Run Value            Unit
        Type
------------------------------ -------------------- -----------
        -------------------  -------------------  ------------------
--
        --------------------
kernel mode                     threaded             0
              process        threaded             not applicable
        static

(1 row affected)
Configuration option changed. Since the option is static, SAP ASE
must be rebooted in order for the change to take effect. Changing the
value of 'kernel mode' does not increase the amount of memory SAP ASE
uses
```

You must restart SAP ASE for the change to take affect.

Consider that:

- When you switch from threaded to process mode, SAP ASE ignores thread pool information in the configuration file when it starts.
- When switching from process kernel mode to threaded mode, all execution classes lose their engine groups and are associated with `syb_default_pool`. The administrator may then associate the execution classes with engine thread pools. When switching from threaded mode to process mode, execution classes lose their thread pool association and are associated with the existing ANYENGINE engine group. The administrator may then associate the execution classes with engine groups. The reassignment of an execution class to a thread pool (or engine group) applies to existing connections associated with that execution class in addition to new logins
- Engine groups are obsolete when running in threaded mode. In this mode, use **sp_addexeclass** to bind execution classes to user thread pools. Thread pools separate processor resources between applications within SAP ASE.

Tasks

A task is an executable that is internal to SAP ASE (for example, user connections, daemons such as the housekeeper task, and kernel tasks such as I/O handling).

SAP ASE schedules tasks to threads, which assign CPU time to each task. SAP ASE tasks include:

- User tasks – represent user connections and are capable of executing user queries. An **isql** connection is an example of a user task.
- Service tasks – not associated with specific users, and do not issue user queries. Service tasks include the housekeeper tasks, checkpoints, replication agent threads, and so on).
- System tasks – kernel-level versions of service tasks. System tasks include I/O handlers, the clock handler, cluster membership service, IP link monitor, and so on.

SAP ASE multiplexes tasks across the threads. Run-to-completion (RTC) thread pools place a task in a thread pool and the next available thread picks up the task, which remains on the thread until it completes, and terminates when it finishes running.

See *Performance and Tuning Series: Basics > Base Priority* for a description of setting the task priority for an individual task.

The `monTask` monitoring table includes a row for each task running on SAP ASE. See the *Reference Manual: Tables*.

Using Threads to Run Tasks

SAP ASE assigns tasks to thread pools, and all thread pools have threads. Each thread pool includes a scheduler that assigns tasks to threads.

The scheduler assigns—and unassigns—muliplexed tasks to threads as they perform work. The scheduler assignes RTC tasks to a thread once and it stays with that thread until the work is complete.

SAP ASE contains both system-created and user-created thread pools. All system-defined thread pools start with the `syb_` prefix (for example `syb_default_pool`). Names for user-defined thread pools cannot start with the `syb_` prefix and must follow the naming conventions for objects (See "SQL Building Blocks," in the *Reference Manual: Blocks*). By default, SAP ASE associates user tasks with the `syb_default_pool` thread pool. Use **sp_bindexeclass** to associate one or more tasks with a user-created thread pool.

Configuring an SMP Environment

Configuring the SMP environment is similar to configuring a uniprocessor environment, although SMP machines are typically more powerful and handle many more users. In an SMP environment, you can also control the number of engines.

Thread Pools

Thread pools group CPU resources, and contain threads used to execute SAP ASE tasks associated with that thread pool.

Threads host engines that execute user tasks, run specific jobs (such as signal handling), and process requests from a work queue. SAP ASE contains system-defined thread pools and, if present, user-created thread pools.

Note: Thread pools are available only when SAP ASE is configured for threaded mode.

Thread pools that contain engines are called engine pools, and perform SAP ASE tasks that have a kernel process ID (KPID). SAP ASE assigns an engine to each thread in an engine pool.

SAP ASE supports these thread types:

- Engine (or multiplexed) threads – execute database query processes, and may be shared between multiple database processes. Multiplexed threads run tasks that share one or more threads with other tasks. By default, user and service tasks are multiplexed. Multiplexed tasks must yield after consuming a defined timeslice, or when blocked. Because all multiplexed threads are assigned an engine, they are equivalent to engines in process mode.
- Run to completion (RTC) threads – used by system tasks and are not shared between multple tasks. An RTC thread runs a single task until the task completes, and is not subject to Adapative Server scheduling.

  RTC threads run tasks that cannot remove themselves from a thread's schedule until the thread completes, do not yield for timeslices, and remain connected to the thread while blocked. RTC tasks may need to wait for a thread to execute if all RTC threads are currently running tasks.

You cannot designate a thread's type when you create the thread pool. User-created thread pools are always multiplexed.

SAP ASE includes these system-defined thread pools:

- `syb_default_pool` – the default engine thread pool. Each thread in `syb_default_pool` is an engine. All user tasks and all multiplexed system tasks (such as the housekeeper) run in `syb_default_pool`. However, you can move some tasks out of `syb_default_pool` by creating additional thread pools.

- `syb_system_pool` – an RTC thread pool used for system threads. Each thread in `syb_system_pool` is dedicated to running a specific task. `syb_system_pool` contains at least one thread for the system clock and other asynchronous signals. All I/O handling threads run in `syb_system_pool`.
- `syb_blocking_pool` – an RTC pool SAP ASE uses to process blocking call requests from multiplexed tasks, which are normally operating system calls that may cause a multiplexed—or engine—thread to block for an unacceptable amount of time. Threads in `syb_blocking_pool` typically consume very few CPU resources.

Thread pools are defined by their attributes, some of which are automatically assigned by SAP ASE, and that others are determined when you create the thread pool. The thread pool attributes are:

- `ID` – a thread pool's system-assigned ID. SAP ASE may assign new IDs to thread pools during start-up, so SAP recommends that you do not make static references to thread pool IDs.
- `Name` – thread pool name. Only system thread pools can start with the `syb_` prefix.
- `Description` – (Optional) thread pool description, up to 255 characters.
- `Type` – is one of `Engine` or `RTC`.
- `Current number of threads` – the number of threads the pool currently contains (may differ for a short period of time from the configured number of threads in the pool while the pool is changing sizes).
- `Configured number of threads` – the number of threads for which the thread pool is configured.
- `idle timeout` – the amount of time, in microseconds, after a thread becomes idle before it goes to sleep.

SAP ASE records the thread pool configuration in the configuration file. This example shows the three default thread pools (`syb_blocking_pool`, `syb_system_pool`, and `syb_default_pool`) and a user-created thread pool named `big_pool`:

```
[Thread Pool:big_pool]
     description = Big thread pool
     number of threads = 15

[Thread Pool:syb_blocking_pool]
     number of threads = 20

[Thread Pool:syb_default_pool]
      number of threads = 1
```

Use **sp_helpthread**, or the `monThreadPool` monitoring table, to view the current thread pool configuration. Edit the thread pool information in the configuration file before starting SAP ASE or use **alter thread pool** to a change the thread pool configuration.

See *System Administration Guide: Volume 2 > Configuring Memory*.

Dynamic Thread Assignment

SAP ASE uses dynamic and static thread assignments.

Dynamic thread assignment allows SAP ASE to execute parallel query plans faster and with fewer resources than using static threads. SAP ASE applies dynamic thread assignment to parallel lava query plans that are generated for **select** queries. However, these commands use static thread assignment:

- **select into**
- **reorg** commands that include a data copy
- **alter table** commands that include a data copy
- **create index**

Dynamic thread assignment improves performance by:

- Executing query plans in parallel with fewer threads in situations where static thread assignment requires recompilation into a serial query plan.
- Executing dynamic load balancing between worker threads: When there are fewer threads executing the query plan than there are work units, the threads that execute the smallest work units complete their task more quickly, freeing themselves to execute the remaining available work units while the threads executing larger work units complete their tasks.
- Using existing semantic partitioning in joins more effectively. Dynamic thread assignment allows a partition-to-partition join when joining two tables that are already partitioned on the joining column, as opposed to static thread assignment, which joins all the partitions of the outer table to all of the partitions of the inner table in one operation. Dynamic thread assignment allows a single worker thread to join the first partition of the outer table to the first partition of the inner table, then reexecutes the query plan fragment to join the second partition, and so on.

Dynamic thread assignment is enabled when you configure SAP ASE for parallelism with the **number of worker processes** and **max parallel degree** parameters. See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Managing Engines

To achieve optimum performance from an SMP system, you must maintain the correct number of engines.

An engine represents a certain amount of CPU power. It is a configurable resource like memory.

Note: SAP ASE uses a load-balancing algorithm to evenly distribute the load among the engines. In process mode, if your server connections use Component Integration Services (CIS), they are affinitied to a single engine, and cannot migrate from one engine to another. This limitation does not apply to threaded mode.

Configuring Engines in Process Mode

When you first install SAP ASE, the system is configured for a single engine. To use multiple engines, reset the number of engines the first time you restart the server. You may also want to reset the number of engines at other times.

For example, you might want to:

- Increase the number of engines if current performance is not adequate for an application and there are enough CPUs on the machine.
- Decrease the number of engines if SAP ASE is not fully utilizing the existing engines. There is overhead involved in running the engines, and SAP ASE is wasting resources if you do not need the extra engines.

Use the **number of engines at startup** configuration parameter to reset the number of engines.

Configuring Engines in Threaded Mode

Edit the thread pool information in the configuration file to administer thread pools, or use **create thread pool**, **alter thread pool**, and **drop thread pool** .

max online engines controls the total number of engines available to SAP ASE.

Use **sp_configure** to reset **max online engines**.

For example, to set the number of engines to 3, issue:

```
sp_configure "max online engines", 3
```

Restart the server to reset the number of engines.

See also

- *Configuring Thread Pools* on page 41

Choosing the Right Number of Engines

It is important that you choose the correct number of engines for SAP ASE.

When choosing the number of engines, consider the following:

- You cannot configure more engines than CPUs. If a CPU goes offline, you may need to decrease the number of engines by using **sp_engine** (in process mode) or **alter thread pool** (in threaded mode).
- SAP ASE may use additional operating system threads to handle I/O operations in threaded mode. In high I/O situations, you must ensure that SAP ASE has sufficient CPU for these threads to run. For example, an SAP ASE performing high I/O on an 8 CPU system may achieve maximum performance with 8 engines in process mode, but only require 7 engines in threaded mode. You may experience severe performance degradation if I/O threads compete with engine threads for CPU time.

- Have only as many engines as you have usable CPUs. If there is a lot of processing by the client or other non-SAP ASE processes, one engine per CPU may be excessive. The operating system may take up part of one of the CPUs.
- Have enough engines. Start with a few engines and add engines when the existing CPUs are almost fully used. If there are too few engines, the capacity of the existing engines is exceeded and may result in bottlenecks.

SAP recommends that you have only as many engines as your site requires. More engines do not necessarily translate into better performance. Generally, SAP ASE perform better with 4 engines that are 80% busy instead of performing with 8 engines that are 40% busy.

Starting and Stopping Engines

Use **sp_engine** to start and stop SAP ASE engines.

Note: SAP ASE must be configured for process mode to use **sp_engine**. When SAP ASE is configured for kernel mode, use **create**, **alter**, and **drop thread** to configure memory pools.

See also

- *Configuring Thread Pools* on page 41

Monitoring Engine Status

Before you bring an engine online or offline, check the status of the engines that are currently running. `sysengines` includes any of the following in the `status` column:

- `online` – indicates the engine is online.
- `in offline` – indicates that **sp_engine offline** has been run. The engine is still allocated to the server, but is having its tasks migrated to other engines.
- `in destroy` – indicates that all tasks have successfully migrated off the engine, and that the server is waiting on the OS-level task to deallocate the engine.
- `in create` – indicates that an engine is being brought online.
- `dormant` – indicates that **sp_engine offline** was not able to migrate all tasks from that engine. If the tasks terminate themselves (through a timeout), engines switch to being permanently offline. Dormant engines process only those tasks that are causing the dormant state; they are not available to work on any other tasks.

The following command shows the engine number, status, number of tasks affinitied, and the time an engine was brought online:

```
select engine, status, affinitied, starttime
from sysengines
```

```
engine status       affinitied  starttime
------ ------------ ----------- --------------------------
     0 online                12        Mar  5 2007 9:40PM
     1 online                 9        Mar  5 2007 9:41PM
     2 online                12        Mar  5 2007 9:41PM
     3 online                14        Mar  5 2007 9:51PM
```

```
     4 online                  8         Mar  5 2007 9:51PM
     5 in offline             10         Mar  5 2007 9:51PM
```

Starting and Stopping Engines with sp_engine

You can dynamically stop or start engines using **sp_engine**, which allows a you to reconfigure CPU resources as processing requirements fluctuate over time.

The syntax for **sp_engine** is:

```
sp_engine {"online" | [offline | can_offline] [, engine_id] |
["shutdown", engine_id]
```

For example, the following brings engine 1 online. Messages are platform-specific (in this example, Sun Solaris was used):

```
sp_engine "online", 1
```

```
02:00000:00000:2001/10/26 08:53:40.61 kernel  Network and device
connection limit is 3042.
02:00000:00000:2001/10/26 08:53:40.61 kernel  SSL Plus security
modules loaded successfully.
02:00000:00000:2001/10/26 08:53:40.67 kernel  engine 1, os pid
8624  online
02:00000:00000:2001/10/26 08:53:40.67 kernel  Enabling Sun Kernel
asynchronous disk I/O strategy
00:00000:00000:2001/10/26 08:53:40.70 kernel  ncheck: Network
fc0330c8 online
```

Use **can_offline** to check whether or not a specific engine can be brought offline. For example, to chech whether engine 1 can be brought offline, use:

```
sp_engine can_offline, 1
```

sp_engine specifies a return code of 0 if you can bring the specified engine offline. If you do not specify an *engine_id*, **sp_engine** describes the status of the engine in `sysengines` with the highest *engine_id*.

You can bring engines online only if **max online engines** is greater than the current number of engines with an **online** status, and if enough CPU is available to support the additional engine.

To bring an engine offline, enter the engine ID. For example, to take engine 1 offline, use:

```
sp_engine offline, 1
```

SAP ASE waits for any tasks that are associated with this engine to finish before taking the engine offline, and returns a message similar to:

```
01:00000:00000:2001/11/09 16:11:11.85 kernel  Engine 1 waiting for
affinitied process(es) before going offline
00:00000:00000:2001/11/09 16:16:01.90 kernel  engine 1, os pid
21127  offline
```

You cannot take engine zero offline.

sp_engine "shutdown" forces any tasks associated with the specified engine to finish in a five-second period, and then shuts down the engine. You can use **sp_engine shutdown** when

an engine has gone into a dormant state or to bring an engine offline. **sp_engine** kills any remaining processes that are preventing the engine from going offline normally. The following shuts down engine 1:

```
sp_engine "shutdown", 1
```

See the *Reference Manual: Procedures*.

Relationship Between Network Connections and Engines

(Process mode only) Due to the operating system limit on the number of file descriptors per process on UNIX, reducing the number of engines reduces the number of network connections that the server can have. On Windows, the number of network connections is independent of the number of engines.

There is no way to migrate a network connection created for server-to-server remote procedure calls—or example, connections to Replication Server and XP Server—so you cannot take an engine offline that is managing one of these connections.

Logical Process Management and dbcc engine(offline)

If you are using logical process management to bind particular logins or applications to engine groups, use **dbcc engine(offline)** carefully.

If you take all engines for an engine group offline:

- The login or application can run on any engine
- An advisory message is sent to the connection logging in to the server

Since engine affinity is assigned when a client logs in, users who are already logged in are not migrated if the engines in the engine group are brought online again with **dbcc engine("online")**.

Managing User Connections

(Process mode only) During process mode, if the SMP system supports network affinity migration (the process of moving network I/O from one engine to another.

SMP systems that support this migration allow SAP ASE to distribute the network I/O load among all of its engines), each engine handles the network I/O for its connections. During login, SAP ASE migrates the client connection task from engine 0 to the engine currently servicing the smallest number of connections. The client's tasks run network I/O on that engine (network affinity) until the connection is terminated. To determine if your SMP system supports this migration, see the configuration documentation for your platform.

By distributing the network I/O among its engines, SAP ASE can handle more user connections. The per-process limit on the maximum number of open file descriptors no longer limits the number of connections. Adding more engines linearly increases the maximum number of file descriptors, as stored in the global variable *@@max_connections*.

As you increase the number of engines, SAP ASE prints the increased *@@max_connections* value to standard output and the error log file after you restart the server. You can query the value using:

```
select @@max_connections
```

This number represents the maximum number of file descriptors allowed by the operating system for your process, minus these file descriptors used by SAP ASE:

- One for each master network listener on engine 0 (one for every "master" line in the `interfaces` file entry for that SAP ASE)
- One for each engine's standard output
- One for each engine's error log file
- Two for each engine's network affinity migration channel
- One per engine for configuration
- One per engine for the `interfaces` file

For example, if SAP ASE is configured for one engine, and the value of *@@max_connections* equals 1019, adding a second engine increases the value of *@@max_connections* to 2039 (if there is only one master network listener).

You can configure the **number of user connections** parameter to take advantage of an increased *@@max_connections* limit. However, each time you decrease the number of engines using **max online engines**, you must also adjust the **number of user connections** value accordingly. Reconfiguring **max online engines** or **number of user connections** is not dynamic, so you must restart the server to change these configuration values. See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Note: Because threaded mode runs a single process, the per-process descriptor limit limits the number of user connections that SAP ASE can support. You may need to adjust the number of file descriptors available in the shell that starts SAP ASE, which may require assistance from the operating system administrator to increase the hard limit.

Configuration Parameters That Affect SMP Systems

Some configuration parameters, such as spinlock ratios, apply only to SMP systems.

See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Configuring Spinlock Ratio Parameters

Spinlock ratio parameters specify the number of internal system resources such as rows in an internal table, or cache, that are protected by one *spinlock*.

A spinlock is a simple locking mechanism that prevents a process from accessing the system resource currently used by another process. All processes trying to access the resource must wait (or "spin") until the lock is released.

Spinlock ratio configuration parameters are meaningful only in multiprocessing systems. An SAP ASE configured with only one engine has only one spinlock, regardless of the value specified for a spinlock ratio configuration parameter.

This table lists system resources protected by spinlocks and the configuration parameters you can use to change the default spinlock ratio.

Configuration Parameter	System Resource Protected
lock spinlock ratio	Number of lock hash buckets
open index hash spinlock ratio	Index metadata descriptor hash tables
open index spinlock ratio	Index metadata descriptors
open object spinlock ratio	Object metadata descriptors
partition spinlock ratio	Rows in the internal partition caches
user log cache spinlock ratio	User log caches

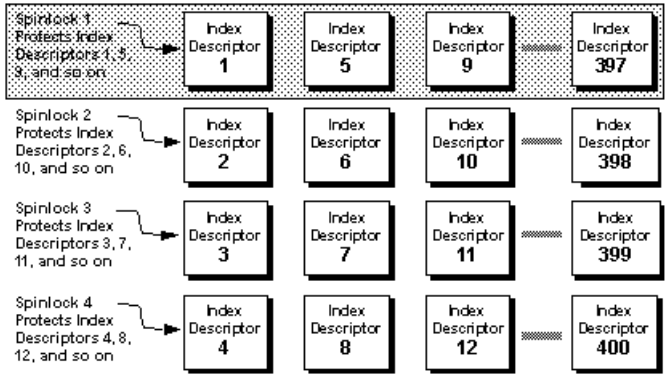
The value specified for a spinlock ratio parameter defines the ratio of the particular resource to spinlocks, not the number of spinlocks. For example, if 100 is specified for the spinlock ratio, SAP ASE allocates one spinlock for each 100 resources. The number of spinlocks allocated by SAP ASE depends on the total number of resources as well as on the ratio specified. The lower the value specified for the spinlock ratio, the higher the number of spinlocks.

Spinlocks are assigned to system resources either in a round-robin manner, or sequentially:

Round-Robin Assignment

Metadata cache spinlocks (configured by the **open index hash spinlock ratio**, **open index spinlock ratio**, and **open object spinlock ratio** parameters) use the round-robin assignment method.

This figure shows one example of the round-robin assignment method and shows the relationship between spinlocks and index metadata descriptors.



Suppose there are 400 index metadata descriptors, or 400 rows, in the index descriptors internal table. You have set the ratio to 100. This means that there are 4 spinlocks: Spinlock 1 protects row 1; Spinlock 2 protects row 2, Spinlock 3 protects row 3, and Spinlock 4 protects row 4. After that, Spinlock 1 protects the next available index descriptor, Index Descriptor 5, until every index descriptor is protected by a spinlock. This round-robin method of descriptor assignment reduces the chances of spinlock contention.

Sequential Assignment

Table-lock spinlocks, configured by the **table lock spinlock ratio** parameter, use the sequential assignment method.

The default configuration for table lock spinlock ratio is 20, which assigns 20 rows in an internal hash table to each spinlock. The rows are divided up sequentially: the first spinlock protects the first 20 rows, the second spinlock protects the second 20 rows, and so on.

In theory, protecting one resource with one spinlock provides the least contention for a spinlock and results in the highest concurrency. In most cases, the default value for these spinlock ratios is probably best for your system. Change the ratio only if there is spinlock contention.

Use **sp_sysmon** to get a report on spinlock contention. See *Performance and Tuning Series: Monitoring SAP Adaptive Server with sp_sysmon*.

CHAPTER 6 Creating and Managing User Databases

System administrators use a number of commands and stored procedures to create and manage databases.

This table summarizes the commands for creating, modifying, and dropping user databases and their transaction logs.

Command	Task
create database...on *dev_name* or **alter database...on** *dev_name*	Makes database devices available to a particular SAP ASE database. When used without the **on** *dev_name* clause, these commands allocate space from the default pool of database devices.
dbcc checktable(syslogs)	Reports the size of the log.
sp_logdevice	Specifies a device that will store the log when the current log device becomes full.
sp_helpdb	Reports information about a database's size and devices.
sp_spaceused	Reports a summary of the amount of storage space used by a database.

See also

- *System-Defined Segments* on page 193

Permissions for Managing User Databases

By default, only the system administrator has **create database** permission, although he or she can grant permission to use the **create database** command.

However, in many installations, to centralize control of database placement and database device allocation, the system administrator maintains a monopoly on **create database** permission. In these situations, the system administrator creates new databases on behalf of other users, and then transfers ownership to the appropriate users.

1. Issues the **create database** command.
2. Switches to the new database with the **use** *database* command.

3. Executes **sp_changedbowner**, as described in *Changing Database Ownership*.

When a system administrator grants permission to create databases, the user that receives the permission must also be a valid user of the `master` database, since all databases are created while using `master`.

The fact that system administrators seem to operate outside the protection system serves as a safety precaution. For example, if a database owner forgets his or her password or accidentally deletes all entries in `sysusers`, a system administrator can repair the damage using the backups or dumps that are made regularly.

Permission for **alter database** or **drop database** defaults to the database owner, and permission is automatically transferred with database ownership. You cannot use grant or revoke to change **alter database** and **drop database** permission.

Using the create database Command

You must have `create database` permission, and you must be a valid user of `master` to use the **create database** command.

Always enter the command **use master** before you create a new database. See the *Reference Manual: Commands*.

Note: Each time you enter the **create database** command, dump the `master` database. This makes recovery easier and safer in case `master` is later damaged.

You can create only one database at a time.

In its simplest form, **create database** creates a database on the default database devices listed in `master..sysdevices`:

```
create database newpubs
```

When a user with the required permission issues **create database**, SAP ASE:

- Verifies that the database name specified is unique and follows the rules for identifiers.
- Makes sure that the specified database device names are available.
- Finds an unused identification number for the new database.
- Assigns space to the database on the specified database devices and updates `master..sysusages` to reflect these assignments.
- Inserts a row into `sysdatabases`.
- Makes a copy of the `model` database in the new database space, thereby creating the new database's system tables.
- Clears all the remaining pages in the database device. If you are creating a database to load a database dump, **for load** skips page clearing, which is performed after the load completes.

The new database initially contains a set of system tables with entries that describe the system tables themselves. It inherits all the changes you have made to the `model` database, including:

- The addition of user names.
- The addition of objects.
- The database option settings. Originally, the options are set to off in `model`. If you want all of your databases to inherit particular options, use **sp_dboption** to change the options in the `model` database. See *System Administration Guide: Volume 1 > System and Optional Databases* and *Setting Database Options*.

After creating a new database, the system administrator or database owner can use **sp_adduser** to manually add users to the database with **sp_adduser**. If you are adding new SAP ASE logins, you may also need the system security officer. See *Security Administration Guide > Manage SAP ASE Logins and Database Users*.

Assigning Space and Devices to Databases

SAP ASE allocates storage space to databases when a user executes the **create database** or **alter database** command.

create database can specify one or more database devices, along with the amount of space on each that is to be allocated to the new database.

Warning! Unless you are creating a small or noncritical database, always place the log on a separate database device. Follow the instructions in "Placing the Transaction Log on a Separate Device" on page 150 to create production databases.

If you use the **default** keyword, or if you omit the **on** clause, SAP ASE puts the database on one or more of the default database devices specified in `master..sysdevices`. See *System Administration Guide: Volume 1 > Initializing Database Devices*.

To specify a size (4MB in the following example) for a database that is to be stored in a default location, use:

```
create database newpubs
on default = "4M"
```

To place the database on specific database devices, include the names of the database devices in the command. You can request that a database be stored on more than one database device, with a different amount of space on each. All the database devices named in **create database** must be listed in `sysdevices` (that is, they must have been initialized with **disk init**). See *System Administration Guide: Volume 1 > Initializing Database Devices*

The following statement creates the `newdb` database and allocates 3MB on `mydata` and 2MB on `newdata`. The database and transaction log are not separated:

```
create database newdb
on mydata = "3M", newdata = "2M"
```

If the amount of space you request on a specific database device is unavailable, SAP ASE creates the database with as much space as possible on each device and displays a message informing you how much space it has allocated on each database device. If there is less than the minimum space necessary for a database on the specified database device, **create database** fails.

If you create (or alter) a database on a UNIX device file that does not use the **dsync** setting, SAP ASE displays an error message in the error log file, for example:

```
Warning: The database 'newdb' is using an unsafe virtual device
'mydata'. The recovery of this database can not be guaranteed.
```

Default Database Size and Devices

If you omit the **size** parameter in the **on** clause, SAP ASE creates the database with a default amount of space.

This amount is the larger of the sizes specified by the **default database size** configuration parameter and the `model` database.

The smallest database you can create is the size of the `model` database, which is determined by your installation's logical page size. To increase the minimum size of a database, use **alter database** to enlarge the model database. You can also use the **default database size** configuration parameter to determine the default database size. See *System Administration Guide: Volume 1 > Setting Configuration Paramters*.

If you omit the **on** clause, the database is created as the default size, as described above. The space is allocated in alphabetical order by database device name, from the default database devices specified in `master..sysdevices`.

To see the logical names of default database devices, enter:

```
select name
    from sysdevices
    where status & 1 = 1
    order by name
```

sp_helpdevice also displays "default disk" as part of the description of database devices.

Estimating the Required Space

The size allocation decisions you make about devices and databases are important because it is difficult to reclaim storage space after it has been assigned.

You can always add space; however, you cannot deallocate space that has been assigned to a database, unless you drop the database.

You can estimate the size of the tables and indexes for your database by using **sp_estspace** or by calculating the value. See *Performance and Tuning Series: Physical Database Tuning > Determining Sizes of Tables and Indexes*.

Placing a Transaction Log on a Separate Device

Use the **log on** clause of the **create database** command to place a transaction log (the `syslogs` table) on a separate database device.

Unless you are creating very small, noncritical databases, always place the log on a separate database device. Placing the logs on a separate database device:

- Lets you use **dump transaction**, rather than **dump database**, thus saving time and tapes.
- Lets you establish a fixed size for the log to keep it from competing for space with other database activity.
- Creates default free-space threshold monitoring on the log segment and allows you to create additional free-space monitoring on the log and data portions of the database.
- Improves performance.
- Ensures full recovery from hard disk crashes. A special argument to **dump transaction** lets you dump your transaction log, even when your data device is on a damaged disk.

To specify a size and device for the transaction log, use the **log on** *device* **=** *size* clause to **create database**.

The size is in the unit specifiers “k” or “K” (kilobytes), “m” or “M” (megabytes), and “g” or “G” (gigabytes), “t” or “T” (terabytes).

For example, this statement creates the `newdb` database, allocates 8MB on `mydata` and 4MB on `newdata`, and places a 3MB transaction log on a third database device, `tranlog`:

```
create database newdb
on mydata = "8M", newdata = "4M"
log on tranlog = "3M"
```

See also

- *Chapter 19, Managing Free Space with Thresholds* on page 419

Estimating the Transaction Log Size

The size of the transaction log is determined by the amount of update activity in the associated database and the frequency of transaction log dumps.

This is true whether you perform transaction log dumps manually or use threshold procedures to automate the task. As a general rule, allocate to the log 10 to 25 percent of the space that you allocate to the database.

Inserts, deletes, and updates increase the size of the log. **dump transaction** decreases its size by writing committed transactions to disk and removing them from the log. Since **update** statements require logging the “before” and “after” images of a row, applications that update many rows at once should plan on the transaction log being at least twice as large as the

number of rows to be updated at the same time, or twice as large as your largest table. Or you can *batch* the updates in smaller groups, performing transaction dumps between the batches.

In databases that have a lot of insert and update activity, logs can grow very quickly. To determine the required log size, periodically check the size of the log. This also helps you choose thresholds for the log and scheduling the timing of transaction log dumps.

To check the space used by a database's transaction log, first use the database, then enter:

```
dbcc checktable(syslogs)
```

dbcc reports the number of data pages being used by the log. If your log is on a separate device, **dbcc checktable** also tells you how much space is used and how much is free. Here is sample output for a 2MB log:

```
Checking syslogs
The total number of data pages in this table is 199.
*** NOTICE: Space used on the log segment is 0.39 Mbytes, 19.43%.
*** NOTICE: Space free on the log segment is 1.61 Mbytes, 80.57%.
Table has 1661 data rows.
```

To check on the growth of the log, enter:

```
select count(*) from syslogs
```

Repeat either command periodically to see how quickly the log grows.

Default Log Size and Device

If you omit the *size* parameter from the **log on** clause, SAP ASE locates the minimum permitted amount of storage.

If you omit the **log on** clause entirely, SAP ASE places the transaction log on the same database device as the data tables.

Moving the Transaction Log to Another Device

If you did not use the **log on** clause to **create database**, you can move your transaction log to another database device.

sp_logdevice marks the portions of an existing database that exist on a specified device as reserved for the transaction log; it does not move existing data. If your database already has data on this device, SAP ASE does not interpret this data as not being on its proper segment. However, because **dbcc** reports this as an error, no existing part of the log moves to the specified device; the current log data remains where it is until the log has extended onto the new device and you use **dump transaction** to clear that part of the log. Also, **sp_logdevice** does not allocate new space to the database or initialize devices. Instead, it reserves those portions of the specified device for the log that already belong to the database you specify.

The syntax for **sp_logdevice** is:

```
sp_logdevice database_name, devname
```

The database device you name must be initialized with **disk init** and must be allocated to the database with **create** or **alter database**.

1. Execute **sp_logdevice**, naming the new database device.
2. Execute enough transactions to fill the page that is currently in use. The amount of space you need to update depends on the size of your logical pages. You can execute **dbcc checktable(syslogs)** before and after you start updating to determine when a new page is used.
3. Wait for all currently active transactions to finish. You may want to use **sp_dboption** to put the database into single-user mode.
4. Run **dump transaction**, which removes all the log pages that it writes to disk. As long as there are no active transactions in the part of the log on the old device, all of those pages are removed.
5. Run **sp_helplog** to ensure that the complete log is on the new log device.

   Note: When you move a transaction log, the space no longer used by the transaction log becomes available for data. However, you cannot reduce the amount of space allocated to a device by moving the transaction log.
6. If required, run **dbcc findstranded(database_name)** to change the database status from "mixed log and data."

See also

- *Chapter 13, Developing a Backup and Recovery Plan* on page 263

Shrinking Log Space

The **alter database** command includes a **log off** parameter that removes unwanted portions of a database log, allowing you to shrink log space and free storage without re-creating the database.

The syntax is:

```
alter database database_name [log off database_device
    [= size | [from logical_page_number] [to logical_page_number]]
    [, database_device
    [= size | [from logical_page_number] [to logical_page_number]]
```

The parameter may be particularly helpful after running the fully logged option for database operations, such as **select into**, **alter table**, or **reorg rebuild**, when the database ends up with extra allocated space that is no longer needed. See the **dump transaction** command the *Reference Manual: Commands*.

Using dump and load database When Shrinking Log Space

Shrinking log space requires careful planning.

The system administrator should keep the following in mind when performing **dump** and **load database**.

- The database you are loading must have at least as much physical space as when it was dumped.
- The physical space in the database you are loading is apportioned according to the physical fragments in the database that was dumped. This means a "hole"—an allocation unit for which there is no associated physical storage as a result of an earlier **alter database log off** command—in the database to be loaded does not necessarily remain a hole after the load.
- Any leftover space in the database you are loading is apportioned in the same way as it is for **dump** and **load database** without holes.
- You can determine the amount of physical space in the database that was dumped as well as whether it has holes, and information about the size and location of these holes, by running the **load database with headeronly** command.

Shrinking a Log Before a dump and load database

Use the **load database with headeronly** command and **sp_helpdb** system procedure to size the target database before it is loaded with the dump.

The full sequence showing these commands is described in *Example of Sequence Using dump and load database*.

1. Create a database with as many log devices as you need. The example creates two log devices.
2. (Optional) Run **select *** to confirm the database creation and show the device fragments that make up the database.
3. Remove unwanted portions of the log from the database without breaking the database dump sequence by using the **alter database log off** command. If the dump sequence of the database is already broken, **alter database log off** automatically removes any shrunken space from the end of the database. Any space removed that is not at the end of the database always becomes a hole.

   In the example, the shrunken space in the middle of the database has become a hole.
4. The `sysusages` output shows the location and size of the holes (`segmap` = 0 and `location` of 4 in the example). The **sp_helpdb** output shows a summary of the size of the database excluding holes (9MB in the example) and the total size of the holes in the database (3072KB of log-only unavailable space or 3MB in the example):
5. The database to be loaded is 12MB in total, but of this, 9MB are actually physically in the database because of a 3MB hole. A **dump database with headeronly** command verifies

that the database contains 12MB of logical pages and 9MB of physical pages. To load this database, you must create a new database at least 9MB in size.

6. Load the database and bring it online.

In the `sysusages` rows for the newly loaded database, the 9MB of physical space has been rearranged to match that of the dumped database so that the database is now 12MB in size, with only 9MB of physical pages and a 3MB hole.

This example shows the complete sequence you perform when using dump and load database

```
create database sales_db on sales_db_dev=3 log on sales_db_log1=3,
    sales_db_log2=3, sales_db_log1=3, sales_db_log2=3
```

```
00:00:00000:00015:2011/01/21 09:38:28.29 server  Timestamp for
database
'sales_db' is (0x0000,
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_dev'
(1536 logical pages requested).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_log1'
(1536 logical pages requested).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_log2'
(1536 logical pages requested).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_log1'
(1536 logical pages requested).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_log2'
(1536 logical pages requested).
Warning: The database 'sales_db' is using an unsafe virtual device
'sales_db_dev'.
The recovery of this database can not be guaranteed.
Database 'sales_db' is now online.
```

```
select * from sysusages where dbid=4
```

```
dbid segmap lstart size vstart location unreservedpgs  crdate
      vdevno
---- ------ ------ ---- ------ -------- -------------  ------
      ------
    4      3      0 1536      0       0            670  Jan 21 2011
9:38AM     1
    4      4   1536 1536      0       0           1530  Jan 21 2011
9:38AM     2
    4      4   3072 1536      0       0           1530  Jan 21 2011
9:38AM     3
    4      4   4608 1536   1536       0           1530  Jan 21 2011
9:38AM     2
    4      4   6144 1536   1536       0           1530  Jan 21 2011
9:38AM     3

(5 rows affected)
```

```
alter database sales_db log off sales_db_log2
select * from sysusages where dbid=4
```

```
dbid segmap lstart size vstart location unreservedpgs  crdate
      vdevno
---- ------ ------ ---- ------ -------- -------------  ------
      ------
    4     3      0 1536      0        0           670  Jan 21 2011
9:38AM     1
    4     4   1536 1536      0        0          1530  Jan 21 2011
9:38AM     2
    4     0   3072 1536   3072        4          1530  Jan 21 2011
9:38AM    -4
    4     4   4608 1536   1536        0          1530  Jan 21 2011
9:38AM     2

(4 rows affected)
```

```
sp_helpdb sales_db
```

```
 name     db_size        owner dbid  created      durability status
 -------- -------------- -----
----  ------------ ---------- --------------
sales_db  9.0 MB        sa       4  Jan 21, 2011 full      no options
set

(1 row affected)
device_fragments   size    usage      created              free kbytes
------------------ ------  ---------- ------------------    --------
------
sales_db_dev       3.0 MB  data only  Jan 21 2011
9:38AM              1340
sales_db_log1      3.0 MB  log only   Jan 21 2011 9:38AM    not
applicable
sales_db_log1      3.0 MB  log only   Jan 21 2011 9:38AM    not
applicable

-----------------------------------------------------------------
log only free kbytes = 6082, log only unavailable kbytes = 3072
(return status = 0)
```

```
dump database sales_db to "c:/temp/sales_db.dmp"
```

```
Backup Server session id is: 45. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from
the Backup Server.
Backup Server: 4.41.1.1: Creating new disk file c:/temp/
sales_db.dmp.
Backup Server: 6.28.1.1: Dumpfile name 'sales_db11021087C7  ' section
number 1
mounted on disk file 'c:/temp/sales_db.dmp'
Backup Server: 4.188.1.1: Database sales_db: 848 kilobytes (67%)
DUMPED.
Backup Server: 4.188.1.1: Database sales_db: 862 kilobytes (100%)
DUMPED.
Backup Server: 3.43.1.1: Dump phase number 1 completed.
```

```
Backup Server: 3.43.1.1: Dump phase number 2 completed.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.188.1.1: Database sales_db: 870 kilobytes (100%)
DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database sales_db).
```

```
load database sales_db from "c:/temp/sales_db.dmp" with headeronly
```

```
Backup Server session id is: 48. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'sales_db11021087C7  ' section
number 1 mounted
on disk file 'c:/temp/sales_db.dmp'
This is a database dump of database ID 4, name 'sales_db', from Jan
21 2011 9:39AM. ASE
version: lite_642236-1/Adaptive Server Enterprise/15.7/EBF 18567 SMP
Drop#2/B/X64/Windows Server/aseasap/ENG/. Backup Server version:
Backup
Server/15.7/B/X64/Windows Server/aseasap/ENG/64-bit/DEBUG/Thu Jan 20
11:12:51 2011.
Database page size is 2048.
Database contains 6144 pages; checkpoint RID=(Rid pageid = 0x604; row
num = 0x12); next
object ID=560001995; sort order ID=50, status=0; charset ID=2.
Database log version=7; database upgrade version=35; database
durability=UNDEFINED.
segmap: 0x00000003 lstart=0 vstart=[vpgdevno=1 vpvpn=0] lsize=1536
unrsvd=670
segmap: 0x00000004 lstart=1536 vstart=[vpgdevno=2 vpvpn=0]
lsize=1536 unrsvd=1530
Unavailable disk fragment: lstart=3072 lsize=1536
segmap: 0x00000004 lstart=4608 vstart=[vpgdevno=2 vpvpn=1536]
lsize=1536 unrsvd=1530
The database contains 6144 logical pages (12 MB) and 4608 physical
pages (9 MB).
```

```
create database sales_db2 on sales_db_dev=3 log on sales_db_log1=6
```

```
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_dev'
(1536 logical pages requested).
CREATE DATABASE: allocating 3072 logical pages (6.0 megabytes) on
disk 'sales_db_log1'
(3072 logical pages requested).
Warning: The database 'sales_db2' is using an unsafe virtual device
'sales_db_dev'.
The recovery of this database can not be guaranteed.
Database 'sales_db2' is now online.
```

```
select * from sysusages where dbid=db_id("sales_db2")
```

```
dbid segmap lstart size vstart location unreservedpgs crdate
      vdevno
---- ------ ------ ---- ------ -------- ------------ ------
      ------
```

```
   5      3      0 1536   1536        0           670 Jan 26 2011
1:22AM      1
   5      4   1536 3072   3072        0          3060 Jan 26 2011
1:22AM      2
```

```
load database sales_db2 from "/tmp/sales_db.dmp"
```

```
Backup Server session id is: 10. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'sales_db1102602564  ' section
number 1 mounted
on disk file '/tmp/sales_db.dmp'
Backup Server: 4.188.1.1: Database sales_db2: 6148 kilobytes (33%)
LOADED.
Backup Server: 4.188.1.1: Database sales_db2: 9222 kilobytes (50%)
LOADED.
Backup Server: 4.188.1.1: Database sales_db2: 9230 kilobytes (100%)
LOADED.
Backup Server: 3.42.1.1: LOAD is complete (database sales_db2).
Started estimating recovery log boundaries for database 'sales_db2'.
Database 'sales_db2', checkpoint=(1544, 22), first=(1544, 22),
last=(1544, 22).
Completed estimating recovery log boundaries for database
'sales_db2'.
Started ANALYSIS pass for database 'sales_db2'.
Completed ANALYSIS pass for database 'sales_db2'.
00:00:00000:00011:2011/01/26 05:12:15.86 server  Log contains all
committed transactions
until 2011/01/26 01:55:15.71 for database sales_db2.
Started REDO pass for database 'sales_db2'. The total number of log
records to process
is 1.
Completed REDO pass for database 'sales_db2'.
00:00:00000:00011:2011/01/26 05:12:15.88 server  Timestamp for
database 'sales_db2' is
(0x0000, 0x00001612).
Use the ONLINE DATABASE command to bring this database online; ASE
will not bring it
online automatically.
```

```
online database sales_db2
```

```
Started estimating recovery log boundaries for database 'sales_db2'.
Database 'sales_db2', checkpoint=(1544, 22), first=(1544, 22),
last=(1544, 22).
Completed estimating recovery log boundaries for database
'sales_db2'.
Started ANALYSIS pass for database 'sales_db2'.
Completed ANALYSIS pass for database 'sales_db2'.
00:00:00000:00011:2011/01/26 05:12:22.49 server  Log contains all
committed transactions
until 2011/01/26 01:55:15.71 for database sales_db2.
Recovery of database 'sales_db2' will undo incomplete nested top
actions.
Database 'sales_db2' is now online.
```

```
select * from sysusages where dbid=db_id("sales_db2")
```

```
dbid segmap lstart size vstart location unreservedpgs crdate
      vdevno
---- ------ ------ ---- ------ -------- ------------ ------
      ------
   5      3      0 1536   1536        0          670 Jan 26 2011
5:12AM      1
   5      4   1536 1536   3072        0         1530 Jan 26 2011
5:12AM      2
   5      0   3072 1536   3072        4         1530 Jan 26 2011
5:12AM     -5
   5      4   4608 1536   4608        0         1530 Jan 26 2011
5:12AM      2
```

Using dump and load transaction When Shrinking Log Space

The size of a log might change during a dump sequence in which the database is first dumped, and then transaction log dumps are performed periodically.

This is particularly true, for example, if the log segment is increased to accommodate the increased volume of logging that is done by a fully logged **select into**, and then the log is shrunk after completion of the command to return the log to its former size. Use these guidelines to load such a dump sequence:

- Create the database that is being loaded from the dumps with the largest size during the dump sequence. Determine this by executing **dump tran with headeronly** on the last transaction dump to be loaded.
- Shrink the log to the size needed only after the transaction log sequence has completed and you have brought the database online.

This example follows the numbered steps described in *Example Using dump and load transaction*.

Shrinking Log Space

Shrinking a database log involves dumping the database and log, altering the size of the database, and loading the database and log dumps.

1. Create the database. The example creates `sales_db`.
2. Turn on full logging of the database using the **sp_dboption** system procedure's **'full logging for all'** database option.
3. Dump the database.
4. Increase the size of the log segment using **alter database log on** in preparation for the execution of a fully logged **select into** command.
5. Run the fully logged **select into** command that makes use of the increased log segment.
6. Dump the transaction log to truncate the log to prepare for shrinking the log segment.
7. Shrink the database log using **alter database log off** to remove the log space added in the earlier step.

8. Dump the transaction log of the database with the shrunken log segment.
9. Before loading the sequence of dumps, get the logical size of the database from the last file in the load sequence. In the example, the size is16MB.

   Note: The logical size of the database from the last dump in the load sequence is guaranteed to be at least as big as the maximum physical size of the database throughout the dump sequence. This provides a convenient method of determining what size the target database should be to load all the dumps in the sequence.

   Use the **load transaction with headeronly** command to determine the size that the target database must be, in order to accommodate all the dumps in the sequence.
10. Create a new database with as many log devices as you need. The example creates the `sales_db1` database as a 16MB database with two log devices.
11. Load this database.
12. Load transaction logs from the first and second transaction log dumps into the database.
13. Bring the database online.
14. Reduce the size of the database by removing space from its log segment. In the example, the log segment is reduced in size by 10MB.
15. Run **select * from sysusages** to confirm the removal of space from the end of the database. The space that has been removed has become a hole in the database.
16. Use the **with shrink_log** option of **dump database** to remove the hole at the end of the database.
17. Run **select * from sysusages** again to confirm that SAP ASE successfully removed the hole from the end of the database.

This example shows the complete sequence you perform when using **dump** and **load transaction**.

```
create database sales_db on sales_db_dev=3 log on sales_db_log1=3
```

```
00:00:00000:00018:2011/05/05 12:45:06.36 server  Timestamp for
database 'sales_db' is (0x0000, 0x00002aa9).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_dev'
(1536 logical pages requested). CREATE DATABASE: allocating 1536
logical pages (3.0
megabytes) on disk 'sales_db_log1' (1536 logical pages requested).
Warning: The database 'sales_db' is using an unsafe virtual device
'sales_db_dev'. The
recovery of this database can not be guaranteed.
Database 'sales_db' is now online.
```

```
sp_dboption sales_db,'full logging for all',true
```

```
Database option 'full logging for all' turned ON for database
'sales_db'.
Running CHECKPOINT on database 'sales_db' for option 'full logging
for all' to take
```

```
effect.
(return status = 0)
```

```
use master
dump database sales_db to "/tmp/sales_db.dmp"
```

```
Backup Server session id is: 120. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 4.41.1.1: Creating new disk file /tmp/sales_db.dmp.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db11137014BC' section
number 1 mounted on disk file '/tmp/sales_db.dmp'
Backup Server: 4.188.1.1: Database sales_db: 852 kilobytes (100%)
DUMPED.
Backup Server: 3.43.1.1: Dump phase number 1 completed.
Backup Server: 3.43.1.1: Dump phase number 2 completed.
Backup Server: 4.188.1.1: Database sales_db: 856 kilobytes (100%)
DUMPED.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.188.1.1: Database sales_db: 860 kilobytes (100%)
DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database sales_db).
```

```
alter database sales_db log on sales_db_log2=10
```

```
Extending database by 5120 pages (10.0 megabytes) on disk
sales_db_log2
Warning: The database 'sales_db' is using an unsafe virtual device
'sales_db_dev'. The
recovery of this database can not be guaranteed.
Warning: Using ALTER DATABASE to extend the log segment will cause
user thresholds on the
log segment within 128 pages of the last chance threshold to be
disabled.
```

```
use sales_db
select * into bigtab2 from bigtab
```

```
(20000 rows affected)
```

```
dump tran sales_db to "/tmp/sales_db.trn1"
```

```
Backup Server session id is: 9. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 4.41.1.1: Creating new disk file /tmp/sales_db.trn1.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db1113903D37' section
number 1 mounted on
disk file '/tmp/sales_db.trn1'
Backup Server: 4.58.1.1: Database sales_db: 250 kilobytes DUMPED.
Backup Server: 4.58.1.1: Database sales_db: 254 kilobytes DUMPED.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.58.1.1: Database sales_db: 258 kilobytes DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database sales_db).
```

```
use master
alter database sales_db log off sales_db_log2
```

```
Removing 5120 pages (10.0 MB) from disk 'sales_db_log2' in database
'sales_db'.
```

```
dump tran sales_db to "/tmp/sales_db.trn2"
```

```
Backup Server session id is: 11. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 4.41.1.1: Creating new disk file /tmp/sales_db.trn2.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db1113903D87' section
number 1 mounted on
disk file '/tmp/sales_db.trn2'
Backup Server: 4.58.1.1: Database sales_db: 6 kilobytes DUMPED.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.58.1.1: Database sales_db: 10 kilobytes DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database sales_db).
```

```
load tran sales_db from "/tmp/sales_db.trn2" with headeronly
```

```
Backup Server session id is: 13. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db1113903D87' section
number 1 mounted on
disk file '/tmp/sales_db.trn2'
This is a log dump of database ID 5, name 'sales_db', from May 19
2011  4:22AM.
ASE version: lite_670673-1/Adaptive Server Enterprise/15.7.0/EBF
19186 SMP GA
FS3b/B/x86_64/Enterprise Linux/asea. Backup Server version: Backup
Server/15.7/EBF 19186 Drop#3B Prelim/B/Linux AMD Opteron/Enterprise
Linux/aseasap/3556/64-bi. Database page size is 2048.
Log begins on page 1986; checkpoint RID=Rid pageid = 0x7c2; row num =
0x14;
previous BEGIN XACT RID=(Rid pageid = 0x7c2; row num = 0x4); sequence
dates:
(old=May 19 2011  4:21:11:356AM, new=May 19 2011  4:22:31:043AM);
truncation
page=1986; 123 pages deallocated; requires database with 8192 pages.
Database log version=7; database upgrade version=35; database
durability=UNDEFINED.
segmap: 0x00000003 lstart=0 vstart=[vpgdevno=1 vpvpn=0] lsize=1536
unrsvd=192
segmap: 0x00000004 lstart=1536 vstart=[vpgdevno=2 vpvpn=0]
lsize=1536
unrsvd=1530
Unavailable disk fragment: lstart=3072 lsize=5120
The database contains 8192 logical pages (16 MB) and 3072 physical
pages (6MB).
```

```
create database sales_db1 on sales_db_dev=3 log on sales_db_log1=3,
sales_db_log2=10
```

```
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_dev'
(1536 logical pages requested).
CREATE DATABASE: allocating 1536 logical pages (3.0 megabytes) on
disk 'sales_db_log1'
(1536 logical pages requested).
CREATE DATABASE: allocating 5120 logical pages (10.0 megabytes) on
disk 'sales_db_log2'
(5120 logical pages requested).
Warning: The database 'sales_db1' is using an unsafe virtual device
'sales_db_dev'. The
recovery of this database can not be guaranteed.
Database 'sales_db1' is now online.
```

```
load database sales_db1 from "/tmp/sales_db.dmp"
```

```
Backup Backup Server session id is: 15. Use this value when executing
the 'sp_volchanged'
system stored procedure after fulfilling any volume change request
from the Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db111390340B' section
number 1 mounted on
disk file '/tmp/sales_db.dmp'
Backup Server: 4.188.1.1: Database sales_db1: 6148 kilobytes (37%)
LOADED.
Backup Server: 4.188.1.1: Database sales_db1: 6160 kilobytes (100%)
LOADED.
Backup Server: 3.42.1.1: LOAD is complete (database sales_db1).
All dumped pages have been loaded. ASE is now clearing pages above
page 3072, which were
not present in the database just loaded.
ASE has finished clearing database pages.
Started estimating recovery log boundaries for database 'sales_db1'.
Database 'sales_db1', checkpoint=(1863, 13), first=(1863, 13),
last=(1865, 7).
Completed estimating recovery log boundaries for database
'sales_db1'.
Started ANALYSIS pass for database 'sales_db1'.
Completed ANALYSIS pass for database 'sales_db1'.
Started REDO pass for database 'sales_db1'. The total number of log
records to process
is 22.
Redo pass of recovery has processed 2 committed and 0 aborted
transactions.
Completed REDO pass for database 'sales_db1'.
Use the ONLINE DATABASE command to bring this database online; ASE
will not
bring it online automatically.
```

```
load tran sales_db1 from "/tmp/sales_db.trn1"
```

```
Backup Server session id is: 17. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db1113903D37' section
number 1 mounted on
```

```
disk file '/tmp/sales_db.trn1'
Backup Server: 4.58.1.1: Database sales_db1: 250 kilobytes LOADED.
Backup Server: 4.58.1.1: Database sales_db1: 258 kilobytes LOADED.
Backup Server: 3.42.1.1: LOAD is complete (database sales_db1).
Started estimating recovery log boundaries for database 'sales_db1'.
Database 'sales_db1', checkpoint=(1863, 13), first=(1863, 13),
last=(1986, 3).
Completed estimating recovery log boundaries for database
'sales_db1'.
Started ANALYSIS pass for database 'sales_db1'.
Completed ANALYSIS pass for database 'sales_db1'.
Started REDO pass for database 'sales_db1'. The total number of log
records to process
is 365.
Redo pass of recovery has processed 8 committed and 0 aborted
transactions.
Completed REDO pass for database 'sales_db1'.
Use the ONLINE DATABASE command to bring this database online; ASE
will not bring it
online automatically.
```

```
load tran sales_db1 from "/tmp/sales_db.trn2"
```

```
Backup Server session id is: 19. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'ales_db1113903D87' section
number 1 mounted on
disk file '/tmp/sales_db.trn2'
Backup Server: 4.58.1.1: Database sales_db1: 10 kilobytes LOADED.
Backup Server: 3.42.1.1: LOAD is complete (database sales_db1).
Started estimating recovery log boundaries for database 'sales_db1'.
Database 'sales_db1', checkpoint=(1986, 3), first=(1986, 3),
last=(1986, 20).
Completed estimating recovery log boundaries for database
'sales_db1'.
Started ANALYSIS pass for database 'sales_db1'.
Completed ANALYSIS pass for database 'sales_db1'.
Started REDO pass for database 'sales_db1'. The total number of log
records to process
is 16.
Redo pass of recovery has processed 2 committed and 0 aborted
transactions.
Completed REDO pass for database 'sales_db1'.
Use the ONLINE DATABASE command to bring this database online; ASE
will not bring it
online automatically.
```

```
online database sales_db1
```

```
Started estimating recovery log boundaries for database 'sales_db1'.
Database 'sales_db1', checkpoint=(1986, 20), first=(1986, 19),
last=(1986, 20).
Completed estimating recovery log boundaries for database
'sales_db1'.
Started ANALYSIS pass for database 'sales_db1'.
```

```
Completed ANALYSIS pass for database 'sales_db1'.
Recovery of database 'sales_db1' will undo incomplete nested top
actions.
Started UNDO pass for database 'sales_db1'. The total number of log
records to process
is 2.
Undo pass of recovery has processed 1 incomplete transactions.
Completed UNDO pass for database 'sales_db1'.
Database 'sales_db1' is now online.
```

```
alter database sales_db1 log off sales_db_log2
```

```
Removing 5120 pages (10.0 MB) from disk 'sales_db_log2' in database
'sales_db1'.
```

```
select * from sysusages where dbid=db_id("sales_db1")
```

```
 dbid  segmap  lstart   size  vstart location unreservedpgs  crdate
             vdevno
 ----  ------  -------  ----  ------ -------- -------------- -------
----------- ------
    6       3       0   1536    1536        0            192  May 19 2011
4:25AM       1
    6       4    1536   1536    1536        0           1536  May 19 2011
4:25AM       2
    6       0    3072   5120    3072        4           5100  May 19 2011
4:25AM      -6

(3 rows affected)
```

```
dump database sales_db1 to "/tmp/sales_db1.dmp" with shrink_log
```

```
Backup Server session id is: 22. Use this value when executing the
'sp_volchanged' system
stored procedure after fulfilling any volume change request from the
Backup Server.
Backup Server: 4.41.1.1: Creating new disk file /tmp/sales_db1.dmp.
Backup Server: 6.28.1.1: Dumpfile name 'sales_db11113903EC3' section
number 1 mounted
on disk file '/tmp/sales_db1.dmp'
Backup Server: 4.188.1.1: Database sales_db1: 3100 kilobytes (100%)
DUMPED.
Backup Server: 3.43.1.1: Dump phase number 1 completed.
Backup Server: 3.43.1.1: Dump phase number 2 completed.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.188.1.1: Database sales_db1: 3108 kilobytes (100%)
DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database sales_db1).
```

```
select * from sysusages where dbid=db_id("sales_db1")
```

```
 dbid  segmap  lstart   size  vstart location unreservedpgs  crdate
             vdevno
 ----  ------  -------  ----  ------ -------- -------------- -------
------------ ------
    6       3       0   1536    1536        0            192  May 19 2011
4:25AM       3
    6       4    1536   1536    1536        0           1530  May 19 2011
```

```
4:25AM      4

(2 rows affected)
```

See also

- *Using the dump and load Commands* on page 281

Calculating the Transaction Log Growth Rate

Use **sp_logging_rate** to calculate the transaction log growth rate for a specified time period.

sp_logging_rate displays the minumum, maximum, and average rate of transaction log growth, in gigabytes per hour, for the period of time you run the system procedure, providing the result as an averaged sum of the calculations, or as iterative results.

This example displays a summary the log growth for the transaction log over a 24-hour period, calculating the growth in one-hour intervals:

```
sp_logging_rate 'sum', '1,00:00:00', '01:00:00'
=========================
Total Summary Information
=========================
Transaction Log Growth Rate      Min GB/h        Max GB/h        Avg
GB/h
---------------------------   --------------  --------------
--------------
                                  0.000000        1.970084
1.566053
```

Database Recovery with the for load Parameter

When you create a new database, SAP ASE generally clears all unused pages in the database device. Use the **for load** option if you are going to use the database to load from a database dump, either for recovery from media failure or for moving a database from one machine to another. Using **for load** runs a streamlined version of **create database** that skips the page-clearing step, and creates a target database that can be used only for loading a dump.

Clearing the pages can take several seconds or several minutes to complete, depending on the size of the database and the speed of your system.

If you create a database using **for load**, you can run only the following commands in the new database before loading a database dump:

- **alter database...for load**
- **drop database**
- **load database**

When you load a database dump, the new database device allocations for the database must match the usage allocations in the dumped database.

After you load the database dump into the new database, there are no restrictions on the commands you can use.

See also

- *Chapter 14, Backing Up and Restoring User Databases* on page 311

Using the with override Option with create database

The **with overide** option allows machines with limited space to maintain their logs on device fragments that are separate from their data.

SAP does not recommend this practice, but it may be the only option available on machines with limited storage, especially if you must get databases back online following a hard-disk failure.

You can still dump your transaction log, but if you experience a media failure, you cannot access the current log, since it is on the same device as the data. You can recover only to the last transaction log dump; all transactions between that point and the failure time are lost.

In this example, the log and data are on separate fragments of the same logical device:

```
create database littledb
    on diskdev1 = "4M"
    log on diskdev1 = "1M"
    with override
```

The minimum database size you can create is the size of `model`.

Changing Database Ownership

A system administrator might want to create user databases and some of the initial work, then grant ownership of them to another user.

sp_changedbowner changes the ownership of a database, and can be executed only by the system administrator in the database where the ownership is to be changed. The syntax is:

```
sp_changedbowner loginame [, true ]
```

The following example makes the user “albert” the owner of the current database:

```
sp_changedbowner albert
```

The new owner must already have a login name in SAP ASE, but he or she cannot be a user of the database or have an alias in the database. You may have to use **sp_dropuser** or **sp_dropalias** before you can change a database’s ownership (see the *Reference Manual: Procedures*). See *Security Administration Guide > Getting Started with Security Administration in SAP ASE* for more information about changing ownership.

Note: You cannot change ownership of the `master` database; it is always owned by the “sa” login.

See also

- *Altering Databases* on page 138

Altering Databases

When your database or transaction log grows to fill all the space allocated with **create database**, you can use **alter database** to add storage.

You can add space for database objects or the transaction log, or both. You can also use **alter database** to prepare to load a database from backup.

Permission for **alter database** defaults to the database owner, and is automatically transferred with database ownership. . **alter database** permission cannot be changed with **grant** or **revoke**.

Note: alter database for proxy update drops all proxy tables in the proxy database.

See also

- *Changing Database Ownership* on page 137

Using alter database

Use **alter database** to extend a database, and specify where storage space is to be added.

In its simplest form, **alter database** adds the configured default amount of space from the default database devices. If your database separates log and data, the space you add is used only for data. Use **sp_helpdevice** to find names of database devices that are in your default list.

See the *Reference Manual: Commands*.

To add space from a default database device to the `newpubs` database, enter:

```
alter database newpubs
```

The **on** and **log on** clauses operate like the corresponding clauses in **create database**. You can specify space on a default database device or some other database device, and you can name more than one database device. If you use **alter database** to extend the `master` database, you can extend it only on the master device. The minimum increase you can specify is 1MB or one allocation unit, whichever is larger.

If SAP ASE cannot allocate the requested size, it allocates as much as it can on each database device, with a minimum allocation of 256 logical pages per device. When **alter database** completes, it prints messages telling you how much space it allocated; for example:

```
Extending database by 1536 pages on disk pubsdata1
```

Check all messages to make sure the requested amount of space was added.

This command adds 2MB to the space allocated for `newpubs` on `pubsdata1`, 3MB on a new device, `pubsdata2`, and 1MB for the log on `tranlog`:

```
alter database newpubs
on pubsdata1 = "2M", pubsdata2 =" 3M"
log on tranlog
```

Note: Each time you issue the **alter database** command, dump the `master` database.

Use **with override** to create a device fragment containing log space on a device that already contains data or a data fragment on a device already in use for the log. Use this option only when you have no other storage options and when up-to-the-minute recoverability is not critical.

Use **for load** only after using **create database for load** to re-create the space allocation of the database being loaded into the new database from a dump.

See also

- *Chapter 14, Backing Up and Restoring User Databases* on page 311

Using the drop database Command

Use **drop database** to remove a database from SAP ASE.

drop database deletes the database and all the objects in it, as well as:

- Freeing the storage space allocated for the database
- Deleting references to the database from the system tables in the `master` database

Only the database owner can drop a database. You must be in the `master` database to drop a database. You cannot drop a database that is open for reading or writing by a user.

The syntax is:

```
drop database database_name [, database_name]...
```

You can drop more than one database in a single statement; for example:

```
drop database newpubs, newdb
```

You must drop all databases from a database device before you can drop the database device itself. The command to drop a device is **sp_dropdevice**.

After you drop a database, dump the `master` database to ensure recovery in case `master` is damaged.

System Tables That Manage Space Allocation

When you create a database on a database device and allocate a certain amount of space to it, SAP ASE first makes an entry for the new database in `sysdatabases`.

SAP ASE then checks `master..sysdevices` to make sure that the device names specified in **create database** actually exist and are database devices.

If you did not specify database devices, or if you used the **default** option, SAP ASE checks `master..sysdevices` and `master..sysusages` for free space on all devices that can be used for default storage. It performs this check in alphabetical order by device name.

The storage space from which SAP ASE gathers the specified amount of storage need not be contiguous. The database storage space can even be drawn from more than one database device. A database is treated as a logical unit, even if it is stored on more than one database device.

Each piece of storage for a database must be at least one allocation unit. The first page of each allocation unit is the allocation page. It does not contain database rows like the other pages, but contains an array that shows how the rest of the pages are used.

The sysusages Table

Database storage information is listed in `master..sysusages`.

Each row in `master..sysusages` represents a space allocation assigned to a database. Thus, each database has one row in `sysusages` for each time **create database** or **alter database** assigns a fragment of disk space to it.

When you install SAP ASE, `sysusages` contains rows for these databases:

- `master`, with a `dbid` of 1
- The temporary database, `tempdb`, with a `dbid` of 2
- `model`, with a `dbid` of 3
- `sybsystemdb`, with a `dbid` of 31513
- `sybsystemprocs`, with a `dbid` of 31514

If you upgraded SAP ASE from an earlier version, databases `sybsystemdb` and `sybsystemprocs` may have different database IDs.

If you installed auditing, the `sybsecurity` database is `dbid` 5.

As you create new databases, or enlarge existing ones, new rows are added to `sysusages` to represent new database allocations.

Here is what `sysusages` might look like on an SAP ASE that includes the five system databases and one user database. The user database was created with the `log on` option, and was extended once using **alter database**. It has a database ID (`dbid`) of 4:

```
select dbid, segmap, lstart, size, vdevno, vstart from sysusages
order by 1
```

```
dbid      segmap      lstart      size        vdevno      vstart
----      ---------   ----------  --------    --------    -----------
1         7           0           6656        0           4
2         7           0           2048        0           8196
3         7           0           1536        0           6660
4         3           0           5120        2           0
4         4           5120        2560        3           0
4         3           7680        5120        2           5120
31513     7           0           1536        0           10244
31514     7           0            63488       1            0
```

In this example, the `lstart` and `size` columns describe logical pages for which the size may vary from 2KB – 16KB bytes. The `vstart` column describes virtual pages (for which the size is always 2KB). These global variables show page size information:

- *@@maxpagesize* – logical page size
- *@@pagesize* – virtual page size

The following matches the database ID to its name, shows the number of megabytes represented by the `size` column, shows the logical device name for each **vdevno** in the list, and computes the total number of megabytes allocated to each database. The example output shows only the result for `dbid` 4, and the result has been reformatted for readability:

```
select  dbid, db_name(dbid) as 'database name', lstart,
     size / (power(2,20)/@@maxpagesize) as 'MB',
     d.name
from sysusages u, sysdevices d
where u.vdevno = d.vdevno
and d.status & 2 = 2
order by 1
compute sum(size / (power(2,20)/@@maxpagesize)) by dbid
```

```
dbid      database name      lstart       MB           device name
------     --------------     --------     --------     -----------------
----
4          test               0            10           datadev
4          test               5120         5            logdev
4          test               7680         10           datadev

Compute Result:
-----------
25
```

The following describes the changes to the `segmap` values in the `sysusages` table as you add segments. The server in the example initially includes the default databases and a user database named `testdb` (a data-only database), and a log on the `testlog` device, as shown in the following output from the `sysusages` table:

```
select dbid, segmap from master..sysusages where dbid = 6
```

```
 dbid     segmap
------   -----------
```

```
  6           3
  6            4
```

If you add a user segment `newseg` to the test database and create table `abcd` on `newseg` and again select the segment information from `sysusages`:

```
sp_addsegment newseg, testdb, datadev
```

```
create table abcd ( int c1 ) on newseg
```

```
select dbid, segmap from sysusages
where dbid=6
dbid     segmap
------   -----------
6           11
6            4
```

Note that the segment mapping for the user database has changed from a value of 3 to a value of 11, which shows that segment mappings for user databases change when you reconfigure a database.

To determine the status of the segments, run:

```
sp_helpsegment
```

```
 segment     name             status
-------    ---------------- ----------
0           system              0
1           default             1
2           logsegment          0
3           newseg              0
```

The segment `newseg` is not part of the default pool.

If you add another segment, `newseg1`, to the `testdb` database and select the segment information from `sysusages` again, the segment mapping for `newseg` has changed from 11 to 27:

```
sp_addsegment newseg1, testdb, datadev
```

```
select dbid, segmap from sysusages
```

```
 dbid    segmap
------  -----------
6            27
6              4
```

The segmap Column

`segmap` shows the storage that is permitted for the database fragment it represents.

You control the bit values in this mask using stored procedures for segment management. The valid bit's numbers in the mask come from `syssegments` in the local database. (Your "local" database is the database you are currently using: either your default database from login, or the database you most recently used with **use database**.)

SAP ASE supplies three named segments:

- `system`, which is segment 0
- `default`, which is segment 1
- `logsegment`, which is segment 2

Use **sp_addsegment** to create additional segments. If you create segments in the `model` database, these segments exist in all databases you subsequently create. If you create them in any other database, they exist only for that database. Different segment names in different databases can have the same segment number. For example, `newseg1` in database `testdb` and `mysegment` in database `mydb` can both have segment number 4.

The `segmap` column is a bitmask linked to the `segment` column in the user database's `syssegments` table. Since the `logsegment` in each user database is segment 2, and these user databases have their logs on separate devices, `segmap` contains 4 (2^2) for the devices named in the **log on** statement and 3 for the data segment that holds the system segment (2^0 = 1) + default segment (2^1 = 2).

Some possible values for segments containing data or logs are:

Value	Segment
3	Data only (system and default segments)
4	Log only
7	Data and log

Values higher than 7 indicate user-defined segments. The `segmap` column is explained more fully in the segments tutorial section in "Creating and Using Segments."

The query below illustrates the connection between segments in `syssegments` and `segmap` in `master..sysusages`. The query lists the segment mappings for the current database, showing that each segment number in `syssegments` becomes a bit number in `master..sysusages`:

```
select dbid, lstart, segmap, name as 'segment name'
from syssegments s, master..sysusages u
where u.segmap & power(2,s.segment) != 0
and dbid = db_id()
order by 1,2
```

```
dbid   lstart    segmap       segment name
-----   ------    --------     --------------
4       0         3            system
4       0         3            default
4       5120      4            logsegment
4       7680      3            system
4       7680       3            default
```

This example shows that disk fragment for `lstart` value 0 and the fragment for `lstart` value 7680 use segments `system` number 0 and `default` number 1, while the fragment for `lstart` value 5120 uses segment `logsegment` number 2. This database was created using

both the **on** and **log on** clauses of **create database**, and was then extended once using the **on** clause of **alter database**.

Because the `sysusages segmap` uses an `int` datatype, it can contain only 32 bits, so no database can hold more than 32 segments (numbered 0 - 31). Because `segmap` is a signed quantity (that is, it can display both positive and negative numbers), segment 31 is perceived as a very large negative number, so the query above generates an arithmetic overflow when you use it in a database that uses segment 31.

The `lstart`, `size`, and `vstart` Columns

Many tables include the `lstart`, `size`, and `vstart` columns.

- `lstart` column – the starting page number in the database of this allocation unit. Each database starts at logical address 0. If additional allocations have been made for a database, as in the case of `dbid` 7, the `lstart` column reflects this.
- `size` column – the number of contiguous pages that are assigned to the same database. The ending logical address of this portion of the database can be determined by adding the values in `lstart` and `size`.
- `vstart` column – the address where the piece assigned to this database begins.
- `vdevno` – the device in which this database fragment resides.

SAP ASE Support for Replication by Column Value

SAP ASE extends Multi-Path Replication™ by including a new distribution model that provides support for distributing replicated rows based on the data values in one or more columns in a single table.

You must enable multiple scanners before you select distribution by column filter.

Filter Conditions

The SAP ASE RepAgent uses the replication filter to identify only the table rows that meet conditions that you specify in the filter from the specified table.

The filter conditions are:

- Comparison operators (=, <, >, and so on)
- Ranges (between and not between)
- Lists (in and not in)
- Character matches (like and not like)
- Unknown values (is null and is not null)
- Combinations of search conditions (and, or)
- Transact-SQL functions that are in allowed in a **where** clause

Note: Only a subset of the Transact-SQL functions are allowed by replication filters in the **where** clause. Transact-SQL functions that are allowed have return values that are

deterministic, meaning the functions always return the same result when given the same input values. An example of a function that is nondeterministic, and therefore not allowed, is **getdate**, which returns a different result every time it is called.

SAP ASE System Table Storage for Replication Filters

A bound replication filter object is stored in these SAP ASE system tables:

- `sysattributes`
- `sysobjects`
- `syscolumns`
- `sysprocedures`
- `syscomments`
- `sysdepends`

The replication filter is represented as `RF` for the value of `sysobjects` table column type and the `object_type` column of the `sysattributes` table.

Creating and Dropping Replication Filter Table Objects

You can create a replication filters table object on the primary SAP ASE database using the **create replication filter** command. See *Additional Distribution Mode for Multi-Path Replication* in the *Replication Server New Features Guide* for a list of restrictions that apply when creating a replication filter.

You can remove a replication filter using the **drop replication filter** command. RepAgent automatically drops filters that you create on a table if you drop the table that has the filters. You cannot drop a replication filter that is bound to a path while RepAgent is running in filter mode. If you bind a filter to a replication path, Replication Server replicates only the data that satisfies the filter conditions across the path.

Displaying Filter Replication Information and Dependencies

Use procedures **sp_depends**, **sp_columns**, **sp_help**, and **sp_helptext** to show filter replication information and dependencies.

Configure Alternate Replication Paths Between a Primary Database and Replication Server.

The **sp_replication_path** procedure has been updated to support binding, unbinding, and listing of filters.

For more information, see:

Additional Distribution Mode for Multi-Path Replication in the *Replication Server New Features Guide*.

To configure replication filters, see *Replication Server Administration Guide Volume 2 > Performance Tuning > Multi-Path Replication > Parallel Transaction Streams > Distribution by Column Value*.

See **create replication filter**, **drop replication filter**, and **sp_replication_path** in the *Replication Server Reference Manual*.

Getting Information about Database Storage

The **sp_helpdb** and **sp_spaceused** system procedure help determine which database devices are currently allocated to databases and how much space each database uses.

Using sp_helpdb to Find Database Device Names and Options

Use **sp_helpdb** with the database name to find the names of the database devices on which a particular database resides.
For example:

```
sp_helpdb pubs2
```

```
name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa           4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size         usage        created     free kbytes
------------------- ------------ ------------ ----------   -------
-----
master               10.0MB        data and log  Apr 13
2005          1792

pubs_2_dev        10.0MB       data and log  Apr 13 2005        9888

device                  segment
----------------------- ----------------------
master                  default
master                  logsegment
master                  system
pubs_2_dev              default
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_helpdb reports on the size and usage of the devices used by the named database. The `status` column lists the database options. These options are described in *System Administration Guide, Volume 1 > Setting Database Options*.

If you are using the named database, **sp_helpdb** also reports on the segments in the database and the devices named by the segments.

When you use **sp_helpdb** without arguments, it reports information about all databases in SAP ASE:

```
sp_helpdb
```

```
name            db_size  owner dbid   created            status
-------------   -------- ----- ----   ------------       -----------
--------
master         48.0 MB   sa      1   Apr 12, 2005       mixed log and
data
model           8.0 MB   sa      3   Apr 12, 2005       mixed log and
data
pubs2          20.0 MB   sa      6   Apr 12, 2005       select into/
     bulkcopy/pllsort, trunc log on chkpt, mixed log and data
sybsystemdb     8.0 MB   sa      5   Apr 12, 2005       mixed log and
data
sybsystemprocs 112.0 MB   sa      4   Apr
12, 2005       trunc log on chkpt,
     mixed log and data
tempdb          8.0 MB   sa      2   Apr 12, 2005       select into/
     bulkcopy/pllsort, trunc log on chkpt, mixed log and data
```

Checking the Amount of Space Used

sp_spaceused provides a summary of space use, in the database, and by tables with indexes and text and image storage.

Checking Space Used in a Database

To get a summary of the amount of storage space used by a database, execute **sp_spaceused** in the database.

For example:

```
sp_spaceused
```

```
database_name                  database_size
------------------------------ -------------
pubs2                          2.0 MB

reserved      data          index_size      unused
------------- ------------- --------------- --------
1720 KB       536 KB        344 KB          840 KB
```

This table describes the columns in **sp_spaceused** output.

Column	Description
data-base_name	The name of the database being examined.
data-base_size	The amount of space allocated to the database by **create database** or **alter database**.
reserved	The amount of space that has been allocated to all the tables and indexes created in the database. (Space is allocated to database objects inside a database in increments of 1 extent, or 8 pages, at a time.)

Column	Description
data, index_size	The amount of space used by data and indexes.
unused	The amount of space that has been reserved but not yet used by existing tables and indexes.

The sum of the values in the unused, index_size, and data columns should equal the figure in the reserved column. Subtract reserved from database_size to get the amount of unreserved space. This space is available for new or existing objects that grow beyond the space that has been reserved for them.

By running **sp_spaceused** regularly, you can monitor the amount of available database space. For example, if the reserved value is close to the database_size value, it indicates that you are running out of space for new objects. If the unused value is also small, it indicates you are also running out of space for additional data.

Checking Summary Information for a Table

Issue **sp_spaceused** with a table name to view information about a specific table.
For example:

```
sp_spaceused titles
```

```
name   rowtotal reserved  data    index_size unused
------ -------- --------- ------- ---------- -----
titles 18       48 KB     6 KB    4 KB       38 KB
```

The rowtotal column may be different than the results of running **select count(*)** on the table. This is because **sp_spaceused** computes the value with the built-in function **rowcnt**. That function uses values that are stored in the allocation pages. These values are not updated regularly, however, so they can be very different for tables with a lot of activity. **update statistics**, **dbcc checktable**, and **dbcc checkdb** update the rows-per-page estimate, so rowtotal is most accurate after you have run one of these commands.

Run **sp_spaceused** regularly on syslogs, since the transaction log can grow rapidly if there are frequent database modifications. This is particularly a problem if the transaction log is not on a separate device, which means it competes with the rest of the database for space.

Checking Information for a Table and Its Indexes

Include the parameter **1** (one) and the table name to view information about the space used by individual indexes.
For example:

```
sp_spaceused titles, 1
```

```
index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
titleidind           2 KB       32 KB      24 KB
titleind             2 KB       16 KB      14 KB
```

```
name       rowtotal   reserved   data   index_size unused
---------- --------  ---------  ------- ---------- ----------
titles           18      46 KB     6 KB       4 KB      36 KB
```

Space taken up by the `text/image` page storage is reported separately from the space used by the table. The object name for `text` and `image` storage is always "t" plus the table name:

```
sp_spaceused blurbs,1
```

```
index_name            size       reserved   unused
--------------------- ---------- ---------- ----------
blurbs                0 KB       14 KB      12 KB
tblurbs               14 KB      16 KB      2 KB

name       rowtotal reserved    data    index_size unused
---------- -------- ----------- ------- ---------- ----------
blurbs             6        30 KB    2 KB       14 KB      14 KB
```

Querying System Table for Space Usage Information

You may want to write some of your own queries to get information about physical storage. For example, to determine the total number of 2K blocks of storage space that exist on SAP ASE, query `sysdevices`:

```
select sum(convert(numeric(20,0), high - low + 1))
from sysdevices
where status & 2 = 2
```

```
-----------------
             230224
```

In this example, the 2 for the `status` column (line 3) indicates a physical device. `high` is the highest valid 2KB block on the device, so you must add 1 to get the true count from the subtraction (`high - low` in the first line) and convert counts to `numeric(20,0)` to avoid overflow from integer addition in the sum.

CHAPTER 7 Database Mount and Unmount

Use the **mount** and **unmount** commands to perform a number of tasks.

- More easily package proprietary databases; for example, as data files instead of as SQL scripts. The associated actions necessary for running these SQL scripts, such as device and database setup, are eliminated.
- Move a database – when you move a set of databases from a source SAP ASE to a destination SAP ASE, you are physically moving the underlying devices.
- Copy databases without a shutting down SAP ASE. When you copy a database from the command line, you must operate outside of SAP ASE, and use commands like the UNIX **dd** or **ftp** to create a byte-for-byte copy of all pages in a set of one or more databases.

  Run **mount** and **unmount** from the **isql** prompt: the primary SAP ASE is the source, and the secondary SAP ASE is the destination. **quiesce database** also allows a single secondary SAP ASE to act as standby for databases from multiple primaries, since databases from multiple sources can be copied to a single destination.

Note: mount database and **unmount database** are supported in the Cluster Edition. These commands may be aborted if an instance failover recovery takes place while you are using these commands. In this case, the user must re-issue the command when the instance failover recovery is complete.

1. Use **unmount** to remove a database and its devices from a server. A manifest file is created for the database at a location you specify in the command clauses. The manifest file contains information pertinent to the database at the source SAP ASE, such as database devices, server information, and database information.
2. Copy or move the database onto the destination SAP ASE.
3. Use **mount** to add the devices, attributes, and so on for the database.
4. Use **database online** to bring the database up on the destination SAP ASE without restarting the server.

See the *Reference Manual: Commands* for complete documentation of **mount** and **unmount database**.

Warning! For every login that is allowed access to a database on the original SAP ASE, a corresponding login for the same `suid` must exist at the destination SAP ASE.

For permissions to remain unchanged, the login maps at the destination SAP ASE must be identical to those on the source SAP ASE.

See also

- *Manifest File* on page 152

Manifest File

The manifest file is a binary file that contains information about the database, such as database devices, server information, and database information.

The manifest file can be created only if the set of databases that occupy those devices are isolated and self-contained. The manifest file contains:

- Source server information that is server-specific or common to all the databases, such as the version of the source SAP ASE, any upgrade version, page size, character set, sort order, and the date the manifest file was created.
- Device information that is derived from the `sysdevices` catalog. The manifest file contains the information on the first and last virtual pages, the status, and the logical and the physical names of the devices involved.
- Database information that is derived from the `sysdatabases` catalog. The manifest file contains information on the database name, `dbid`, login name of the database administrator (dba), `suid` of the owner, pointer to the transaction log, creation date, status bits from the status fields in `sysdatabases`, date of the last **dump tran**, and the `diskmap` entries of the databases involved.

Warning! Operations that perform character translations of the file contents (such as **ftp**) corrupt the file, unless they are performed in binary mode.

See also

- *Chapter 7, Database Mount and Unmount* on page 151

Operations That Copy and Move Databases

Moving or copying operations occur at the database level, and require activity external to SAP ASE. To move or copy devices and databases, be sure that they are set up on the source SAP ASE using units that support a physical transportation.

For example, if any device is used by more than one database, then all of those databases must be transported in one operation.

When you copy a database, you duplicate a set of databases from the source to a destination by physically copying the underlying devices. You are copying a set of databases from a source SAP ASE to a destination SAP ASE.

The **quiesce database** command lets you include the parameter for creating the manifest file for an external dump. Use a utility or command external to SAP ASE (**tar**, **zip** or the UNIX **dd** command) to move or copy the database to the destination SAP ASE. Data is extracted and placed on devices at the destination SAP ASE using the same external command or utility.

If a device is used for more than one database, you must remove all of the databases on that device in one operation.

Use caution during the initial configuration of the source SAP ASE. The SAP ASE cannot verify whether a device is transportable as a unit. Make sure that the underlying disk that is to be disconnected does not cause a database that is not being moved to lose some of its storage. SAP ASE cannot identify whether a drive is partitioned on a single physical disk; you must move the databases together in one unit.

Warning! mount and **unmount** allow you to identify more than one database for a move operation. However, if a device is used for more than one database, then all of the databases must be moved in one operation. Specify the set of databases being transported. The devices used by these databases cannot be shared with any extraneous database besides the ones specified in the command.

Performance Considerations

Database IDs for the transported databases must be the same on the destination SAP ASE unless you are mounting the database for temporary usage, in which case you must run **checkalloc** to fix the database ID.

If the `dbid` is changed, all stored procedures are marked for recompiling in the database. This increases the time it takes to recover the database at the destination, and delays the first execution of the procedure.

Device Verification

The destination SAP ASE verifies the devices in the manifest file by scanning the device allocation boundaries for each database.

The scan ensures that the devices being mounted correspond to the allocations described in the manifest file, and verifies the `dbid` in the allocation page against that in the manifest file for the first and last allocation pages for each `sysusages` entry.

If a stricter device inspection is necessary, use **with verify** in the **mount** command, which verifies the dbid for all allocation pages in the databases.

Exercise extreme care to ensure you do not mix up copies of the devices.

For example, if you make a database copy made up of copies for disks `dsk1`, `dsk2`, and `dsk3` and in August, you try to mount `dsk1` and `dsk2` copies from a copy of the database made in March, and `dsk3` from a copy made in June, the allocation page check passes even if **with verify** is used in the **mount** command. Recovery fails because the database information is invalid for the version being used.

However, recovery may not fail if `dsk3` is not accessed, which means that the database comes online, but the data may be corrupt.

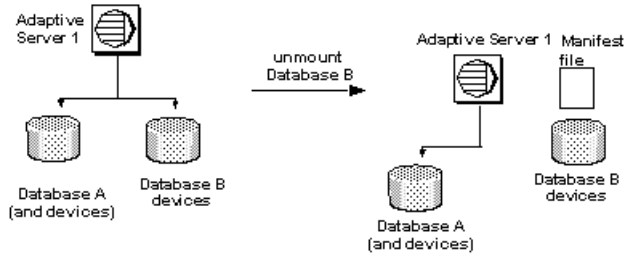
Mounting and Unmounting Databases

The **quiesce database** command includes a clause that facilitates the **mount** and **unmount** commands.

Unmounting a Database

When you unmount a database, you remove the database and its devices from an SAP ASE.

The **unmount** command shuts down the database. All tasks using the database are terminated. The database and its pages are not altered and remain on the operating system devices. This figure shows what happens when you unmount a database from a system.



Note: With **unmount**, you identify more than one database for a move operation. However, if a device is used for more than one database, all of the databases must be moved in one operation. You specify the set of databases being transported. The devices used by these databases cannot be shared with any extraneous database besides the ones specified in the command.

The **unmount** command limits the number of databases that can be moved in a single command to eight.

The **unmount** command:

- Shuts down the database,
- Drops the database from the SAP ASE,
- Deactivates and drops devices,
- Uses the *manifest_ file* clause to create the manifest file.

Once the **unmount** command completes, you can disconnect and move the devices at the source SAP ASE if necessary.

```
unmount database <dbname list> to <manifest_file> [with {override,
[waitfor=<delay time]} ]
```

For example:

```
unmount database pubs2 to "/work2/Devices/Mpubs2_file"
```

If you now try to use the `pubs2` database, you see:

```
Attempt to locate entry in sysdatabases for database 'pubs2' by name
failed - no entry found under that name. Make sure that name is
entered properly.
```

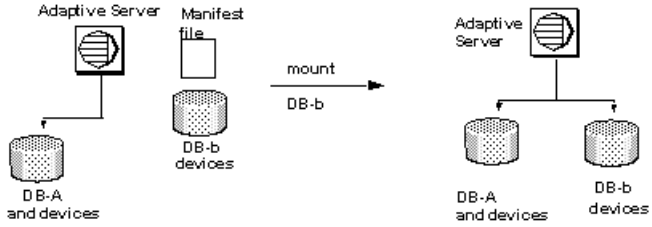
Note: When the referencing database is dropped by the **unmount** command with an override, you cannot drop the referential constraints (dependencies) or table.

Mounting a Database

Use the **mount** command to attach the database to the destination or secondary SAP ASE.

The **mount** command decodes the information in the manifest file and makes the set of databases available online. All the required supporting activities are executed, including adding database devices, if necessary, and activating them, creating the catalog entries for the new databases, recovering them, and putting them online.

The **mount** command limits the number of databases to eight in a single command.



See **mount** in the *Reference Manual: Commands*.

Note: mount allows you to identify more than one database for a move operation. However, if a device is used for more than one database, then all of the databases must be moved in one operation. You specify the set of databases being transported. The devices used by these databases cannot be shared with any extraneous database besides the ones specified in the command.

You can use the **mount** command in different ways:

- Use the **mount** command at the destination SAP ASE. For example:

  ```
  mount database all from "/data/new_user2/mfile1"
  using "/data/new_user1/d0.dbs" = "1dev1"
  ```

  The databases and their devices appear at the destination SAP ASE, marked as in-mount. The system is populated with updates to system catalogs and appropriate information about the databases, but the databases themselves are unrecovered. They do, however, survive a system failure.

The destination SAP ASE then recovers the databases one at a time. The databases remain offline after recovery.

If a recovery fails on a database, it affects only that database. The recovery continues for the other databases.

Use the command **database online** to bring the databases online.

You need not restart the destination server.

- Use the **mount** command with **listonly** to display the path names in the manifest file from the source SAP ASE without mounting the database.

  Before mounting the databases, use **listonly** parameter to list the device path names at the destination SAP ASE. For example:

  ```
  mount database all from "/data/new_user2/mfile1" with listonly

  /data/new_user1/d0.dbs = ldev1
  ```

  Then use **mount** to actually mount the databases. Once you have the path names, verify or modify them to meet your criteria at the destination SAP ASE.

When you **mount** databases onto an SAP ASE:

- You cannot mount a subset of the databases described in the manifest. All the databases and devices present in the manifest must be mounted together.
- The databases being mounted must have the same page size as the previous SAP ASE.
- There must be enough devices configured on the secondary SAP ASE for the successful addition of all the devices belonging to the mounted databases.
- The configuration parameter **number of devices** must be set appropriately.
- Database names and devices with the same names as the mounted database must not already exist.
- SAP ASE must have the same version as the mounted database.
- The mounted database must be from the same platform as the SAP ASE.

To create a mountable version of a database:

1. Use the **quiesce database** command with the manifest clause and quiesce the database. This command creates a `manifest` file describing the database.
2. Use the **mount** command with **listonly** to display the list of devices to be copied.
3. Use external copy utilities, such as **cp, dd, split mirror**, and so on, to copy the database devices to another SAP ASE.The copy of the devices and the manifest file is a mountable copy of the database.

Moving Databases from One SAP ASE to Another

Use the **unmount** command to unmount the database from the first SAP ASE.

1. The **unmount** command creates a manifest file describing the database.

2. Make the database devices available to the second SAP ASE, if not already available. This may require the help of your operating system administrator if the second SAP ASE is on another machine.
3. Execute the **mount** command on the secondary SAP ASE with the manifest file created in step 1.

System Restrictions

mount and **unmount** include restrictions.

Including:

- You cannot unmount system databases. However, you can unmount `sybsystemprocs`.
- You cannot unmount proxy databases cannot be unmounted.
- **mount** and **unmount** database commands are not allowed in a transaction.
- **mount** database is not allowed in an HA-configured server.

quiesce database Extension

Use **quiesce database** with the extension for creating the manifest file to duplicate or copy databases.

quiesce database affects the **quiesce hold** by blocking writes in the database, then creates the manifest file. The command then returns control of the database to the user.

You cannot create a manifest file if the set of databases that are quiesced contain references to databases outside of the set. You may use the override option to bypass this restriction.

Next, use a utility to copy the database to another SAP ASE. You must follow these rules for **quiesce database hold** for the copy operation:

- The copy operation cannot begin until the **quiesce database hold** process has completed.
- Every device for every database in the **quiesce database** command must be copied.
- The copy process must complete before you invoke **quiesce database release**.

See **quiesce database** in the *Reference Manual: Commands*.

CHAPTER 8 Distributed Transaction Management

Distributed transactions can take place in an environment where an external transaction manager coordinates transaction execution using a specific protocol, such as X/Open XA. SAP ASE supports transactions using the CICS, Encina, TUXEDO, and MSDTC transaction managers through the DTM XA interface to SAP ASE.

Note: SAP ASE with the DTM XA interface provides features that were previously part of the XA-Server product. The XA-Server product is not required and is not included with SAP ASE. See the *XA Interface Integration Guide for CICS, Encina, and TUXEDO* for information about the DTM XA interface.

Configuration Considerations

Before configuring SAP ASE for DTM, you should consider how doing so affects the XA interface, external transactions, and RPC, CIS, and SYB2PC transactions.

Behavior for Transaction Manager-Coordinated Transactions

SAP ASE natively implements several features that were part of the XA-Library and XA-Server products, and provides recovery procedures for prepared transactions coordinated via the X/Open XA protocol.

The XA interface to SAP ASE accommodates these distributed transaction management features. Changes to the XA interface are transparent to X/Open XA client applications. However, you must link SAP ASE DTM XA interface to your X/Open XA transaction manager in order to use SAP ASE as a resource manager. Details on all XA interface changes are described in the *XA Interface Integration Guide for CICS, Encina, and TUXEDO*.

SAP ASE also includes support for distributed transactions coordinated by MSDTC. MSDTC clients can communicate directly with SAP ASE using the native interface. Clients can also communicate with one or more SAP ASE running on UNIX by using the DTM XA interface.

Note: MSDTC clients using the DTM XA interface must possess `dtm_tm_role` in the SAP ASE(s) they access. See the *XA Interface Integration Guide for CICS, Encina, and TUXEDO* for more information about `dtm_tm_role`.

Enhanced Transaction Manager for SAP ASE Versions 15.0.3 or Later

In versions of SAP ASE earlier than 15.0.3, when SAP ASE implicitly aborts an external transaction without the application's awareness, DML commands that would normally run inside this transaction might instead be executed outside the explicit transaction.

The DML commands are executed inside an implicit transaction started by SAP ASE. This behavior can result in inconsistent business data. To handle this situation, user applications should always check whether the external transaction is still active, and issue commands accordingly.

In versions 15.0.3 and later, if there is an implicit rollback of the external transaction, SAP ASE does not allow any DML commands to be executed on the connection attached to the external transaction until the transaction manager sends a detach request. The detach request indicates the end of a batch of commands intended for the external transaction.

In versions 15.0.3 and later, Adaptive Serve automatically prevents SQL commands that are intended to execute inside a distributed transaction from executing outside it. The user application no longer has to check the global variable *@@trancount* before every command, to see whether; when a transaction is implicitly aborted, an error message (3953) appears: "Cannot execute the command because the external transaction has been rolled back." This message disappears when a **detach transaction** command is issued.

To suppress the 3953 error messages and let SAP ASE restore the former behavior (that is, executing SQL commands even when the DTM transaction is not active), start SAP ASE using trace flag -T3955.

RPC and CIS Transactions

Local SAP ASE transactions can update data in remote servers by using Transact-SQL remote procedure calls (RPCs) and Component Integration Services (CIS).

RPC updates are accomplished by executing an RPC from within a locally-created transaction. For example:

```
sp_addserver westcoastsrv, ASEnterprise, hqsales
begin transaction rpc_tran1
update sales set commission=300 where salesid="12OZ"
exec westcoastsrv.salesdb..recordsalesproc
commit rpc_tran1
```

This transaction updates the `sales` table on the local SAP ASE, but also updates data on a remote server using the RPC, `recordsalesproc`.

CIS provides a way to update data on remote tables as if those tables were local. By using **sp_addobjectdef** users can create local objects in SAP ASE that reference remote data. Updating the local object modifies data in the remote SAP ASE. For example:

```
sp_addobjectdef salesrec,
"westcoastsrv.salesdb..sales", "table"
begin transaction cis_tran1
```

```
update sales set commission=300 where salesid="12OZ"
update salesrec set commission=300 where salesid="12OZ"
commit cis_tran1
```

SYB2PC Transactions

SYB2PC transactions use the SAP two-phase commit protocol to ensure that the work of a distributed transaction is committed or rolled back as a logical unit.

SAP ASE does not modify the behavior of SYB2PC transactions. However, application developers who implement SYB2PC transactions may want to consider using SAP ASE transaction coordination services instead. Compared to SYB2PC transactions, transactions coordinated directly by SAP ASE use fewer network connections and execute more quickly, while still ensuring the integrity of the distributed transaction. Application code can also be simpler when SAP ASE, rather than the application, coordinates remote transactions.

Enabling DTM Features

The **enable dtm** parameter enables or disables basic DTM features.

When **enable dtm** is set to 1 (on), SAP ASE supports external transactions from MSDTC, and from X/Open XA transaction managers via the DTM XA Interface. See the *XA Interface Integration Guide for CICS, Encina, and TUXEDO* for more information.

To enable basic DTM features, use:

```
sp_configure 'enable dtm', 1
```

You must restart SAP ASE for this change to take effect.

enable xact coordination enables or disables SAP ASE transaction coordination services. When this parameter is enabled, SAP ASE ensures that updates to remote SAP ASE data commit or roll back with the original transaction.

To enable transaction coordination, use:

```
sp_configure 'enable xact coordination', 1
```

You must restart SAP ASE for this change to take effect.

Configuring Transaction Resources

SAP ASE provides a common interface to support both local server transactions and external transactions that are coordinated by distributed transaction protocols.

Distributed transaction protocol support is provided for X/Open XA, MSDTC, and native SAP ASE transaction coordination services.

SAP ASE manages all transactions as configurable server resources, and the system administrator can configure the total number of resources available in a given server. Client

tasks that access SAP ASE in an X/Open XA environment can also suspend and join threads to transaction resources as needed.

This section describes how to determine and configure the total number of transaction resources available to SAP ASE.

Calculating the Number of Required Transaction Descriptors

SAP ASE uses the *transaction descriptor* resource to manage transactions within a server. A transaction descriptor is an internal memory structure that SAP ASE uses to represent a transaction.

Upon starting, SAP ASE allocates a fixed number of transaction descriptors based on the value of the configuration parameter **txn to pss ratio** and places them in a pool. SAP ASE obtains transaction descriptors from the pool as they are needed for new transactions. As transactions complete, descriptors are returned to the pool. If there are no transaction descriptors available, transactions may be delayed as SAP ASE waits for descriptors to become freed.

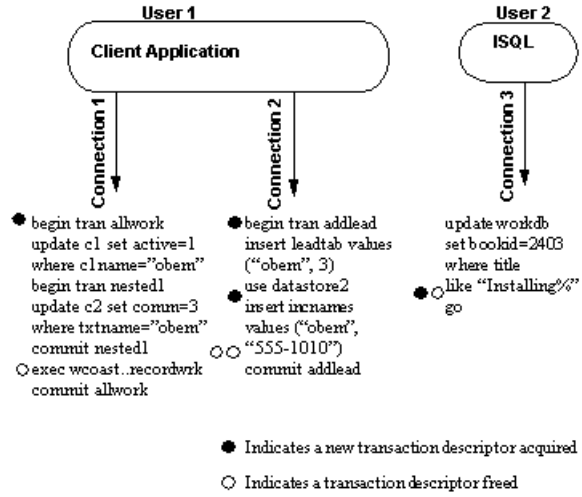
To properly configure the number of transaction descriptors, it is important that you understand exactly when SAP ASE tries to obtain new descriptors from the global pool. A new transaction descriptor is required when:

- A client connection initiates a new, outer-level transaction. This can occur explicitly, when the client executes an outer-level **begin transaction** command. It can also occur implicitly, when a client modifies data without entering a **begin transaction** command.

  Once an outer-level transaction has begun, future nested **begin transaction** commands do not require additional transaction descriptors. Allocation and deallocation of the transaction descriptor is dictated by the outer-most block of the transaction.
- An existing transaction modifies a second database (a multi-database transaction). A multi-database transaction requires a dedicated transaction descriptor for *each* database it accesses.

This figure illustrates how SAP ASE obtains and releases transaction descriptors for different transaction types.

Figure 11: Allocating and Deallocating Transaction Descriptors



In this situation, SAP ASE uses a total of three transaction descriptors for User 1, who accesses the server through a pair of client connections. The server allocates a single descriptor for transaction **allwork**, which is freed when that transaction commits. The nested transaction, **nested1**, does not require a dedicated transaction descriptor.

Transaction **addlead**, a multi-database transaction, requires two transaction descriptors—one for the outer transaction block, and one for modifying a second database, `datastore2`. Both transaction descriptors are released when the outer transaction block commits.

User 2, accessing SAP ASE from **isql**, also requires a dedicated transaction descriptor. Even though User 2 did not explicitly create an outer transaction block with **begin transaction**, SAP ASE implicitly creates a transaction block to execute the **update** command. The transaction descriptor associated with this block is acquired after the **go** command, and released after the insert has completed.

Because transaction descriptors consume memory that can be used by other SAP ASE services, it is important that you use only enough descriptors to satisfy the maximum number of transactions that may be required at any given time.

Setting the Number of Transaction Descriptors

Once you have determined the number of transaction descriptors to use in your system, use **sp_configure** to set the value of **txn to pss ratio**.

txn to pss ratio determines the total number of transaction descriptors available to the server. At start time, this ratio is multiplied by the **number of user connections** parameter to create the transaction descriptor pool:

```
# of transaction descriptors = number of user connections * txn to
pss ratio
```

The default txn to pss ratio value, 16, ensures compatibility with earlier versions of SAP ASE. Prior to version 12.0, SAP ASE allocated 16 transaction descriptors for each user connection. In version 12.0 and later, the number of simultaneous transactions is limited only by the number of transaction descriptors available in the server.

You must restart SAP ASE for this change to take effect.

For example, to allocate 25 transaction descriptors for every user connection, use the command:

```
sp_configure 'txn to pss ratio', 25
```

Using SAP ASE Coordination Services

The work of a local SAP ASE transaction is sometimes distributed to remote servers that modify remote data.

This can happen when a local transaction executes a remote procedure call (RPC) to update data in another SAP ASE table, or when a local transaction modifies data in a remote table using Component Integration Services (CIS).

Overview of Transaction Coordination Services

SAP ASE provides services to propagate transactions to remote servers and coordinate the work of all servers, ensuring that all work is either committed or rolled back as a logical unit.

With these transaction coordination services, SAP ASE itself can act as a distributed transaction manager for transactions that update data in multiple SAP ASEs.

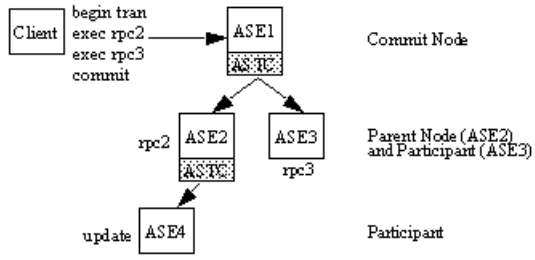
Hierarchical Transaction Coordination

Because other servers involved in a distributed transaction may also coordinate remote participants, transactions can be further propagated to additional servers in a hierarchical manner.

In the figure below, the client connected to ASE1 begins a transaction that executes an RPC on ASE2 and an RPC on ASE3. The coordination service for ASE1 propagates the transaction to ASE2 and ASE3.

Since ASE2 also has transaction coordination services enabled, it can propagate the transaction to additional remote participants. Here, ASE2 propagates the transaction to ASE4 where data is updated using CIS.

Figure 12: Hierarchical Transaction Coordination



Here, ASE1 is referred to as the commit node for the distributed transaction. When the transaction on ASE1 commits, the coordination service for ASE1 instructs ASE2 and ASE3 to prepare the transactions that it propagated to them. ASE3 indicates that its transaction is prepared when its local work is ready to be committed. ASE2 must complete its local work and instruct ASE3 to prepare its transaction. When the transactions are prepared in ASE2 and ASE3, the coordination service in ASE1 commits the original transaction. The instruction to commit subordinate transactions is then transmitted to ASE2, ASE3, and ultimately to ASE3, in the same manner as the instruction to prepare was transmitted.

X/Open XA-Compliant Behavior in DTP Environments

The X/Open XA protocol requires resource managers to provide coordination services for transactions that are propagated to remote resource managers.

This requirement is made because the external transaction manager (and in some cases, the client originating the transaction) has no knowledge of when transactions are propagated to remote servers, and therefore cannot ensure that the remote transactions complete or abort as required.

The new transaction coordination service brings SAP ASE, in its role as a resource manager, into full compliance with the X/Open XA protocol. Distributed transactions can be implicitly propagated to remote servers through RPCs and CIS, and SAP ASE guarantees that the commit or rollback status of the global transaction is preserved in the remote servers it coordinates.

Requirements and Behavior

SAP ASE transaction coordination services can ensure that the work of remote servers is logically committed or rolled back.

Transaction coordination services are transparent to the client executing the distributed transaction. When a local client transaction executes a RPC or updates data via CIS, the coordination service creates a new transaction name for the remote work and propagates that transaction to the subordinate, remote server. When the local client commits or rolls back the local transaction, SAP ASE coordinates that request with each of the subordinate servers to ensure that the remote transactions are committed or rolled back as well.

The SAP ASE transaction coordination service runs as one or more background tasks named "ASTC HANDLER," and can be viewed using **sp_who**. In systems using multiple SAP ASE engines, the number of "ASTC HANDLER" processes (rounded down to the nearest whole number) is:

```
number of engines * 2/3
```

There can be a maximum of 4 "ASTC HANDLER" processes running on SAP ASE.

The following output from **sp_who** shows a single "ASTC HANDLER":

```
sp_who
```

```
fid spid  status   loginame   origname   hostname           blk_spi
d   dbname
   tempdbname   cmd              block_xloid    threadpool
--- ----  --------- ---------  ---------  ----------------   -------
-   -----------
   -----------   ----------------- -----------    -----------------
0     1   running      sa       sa        dtmsol1       0      master
        tempdb             SELECT           0    syb_default_pool
0     2  sleeping     NULL     NULL         master       0      master
        tempdb   NETWORK HANDLER           0    syb_default_pool
0     3  sleeping     NULL     NULL                      0      master
        tempdb     DEADLOCK TUNE           0    syb_default_pool
0     4  sleeping     NULL     NULL                      0      master
        tempdb    MIRROR HANDLER           0    syb_default_pool
0     5  sleeping     NULL     NULL                      0      master
        tempdb       HOUSEKEEPER           0    syb_default_pool
0     6  sleeping     NULL     NULL                      0      master
        tempdb  CHECKPOINT SLEEP           0    syb_default_pool
0     7  sleeping     NULL     NULL     metin1_dtm       0  sybsystemdb
        tempdb      ASTC HANDLER           0    syb_default_pool
```

Ensuring Sufficient Resources for Updates

By default, the transaction coordination service is always enabled.

The system administrator can enable or disable these services using the **enable xact coordination** configuration parameter. See *System Administration Guide: Volume 1 > Setting Configuration Parameters* for a complete description of this parameter.

The system administrator must also ensure that SAP ASE has the required resources to coordinate all of the RPCs and CIS updates that may be requested by transactions. Each time a transaction issues an RPC or CIS update, the transaction coordinator must obtain a free *DTX participant*. A DTX participant or "distributed transaction participant" is an internal memory structure that SAP ASE uses to coordinate a transaction that has been propagated to a subordinate SAP ASE. In the figure above, ASE1 requires three free DTX participants, and ASE2 requires two free DTX participants. (In each case, a single DTX participant is used to coordinate the local work of the transaction that is propagated.

DTX participant resources remain in use by the coordinating SAP ASE until the associated remote transaction has committed. This generally occurs some period of time *after* the

initiating transaction has committed, since the initiating transaction commits as soon as all subordinate transactions have successfully prepared their work.

If no DTX participants are available, RPC requests and CIS update requests cannot proceed and the transaction is aborted.

number of dtx participants Parameter

Configure the total number of DTX participants available in SAP ASE using the **number of dtx participants** configuration parameter.

number of dtx participants sets the total number of remote transactions that the SAP ASE transaction coordination service can propagate and coordinate at one time.

By default, SAP ASE can coordinate 500 remote transactions. Setting **number of dtx participants** to a smaller number reduces the number of remote transactions that the server can manage. If no DTX participants are available, new distributed transactions will be unable to start. In-progress distributed transactions may abort if no DTX participants are available to propagate a new remote transaction.

Setting **number of dtx participants** to a larger number increases the number of remote transaction branches that SAP ASE can handle, but also consumes more memory.

Optimizing the number of dtx participants

During peak periods, use **sp_monitorconfig** to examine the use of DTX participants to verify **number of dtx participants** is configured correctly.
For example:

```
sp_monitorconfig "number of dtx participants"
```

```
Usage information at date and time: Jun 18 1999 9:00AM.
```

```
Name                          Num_Free   Num_Active  Pct_act   Max_Used
      Reuse_cnt    Instance_Name
-----------------------------   --------   ----------  --------  ----
-------
      --------  -------------------------
number of dtx participant
   480           20      4.00           37
         210                        NULL
participants
```

If the `#Free` value is zero or very low, new distributed transactions may be unable to start due to a lack of DTX participants. Consider increasing the **number of dtx participants** value.

If the `#Max Ever Used` value is too low, unused DTX participants may be consuming memory that could be used by other server functions. Consider reducing the value of **number of dtx participants**.

Using Transaction Coordination Services on Remote Servers

When SAP ASE propagates transactions to other SAP ASEs, it can ensure the integrity of the distributed transaction as a whole.

However, the work of a local SAP ASE transaction is sometimes distributed to remote servers that do not support transaction coordination services. This may occur when a transaction uses RPCs to update data in earlier SAP ASE versions, or when CIS services are used to update data in non-SAP databases. Under these circumstances the coordinating SAP ASE cannot ensure that the work of remote servers is committed or rolled back with the original transaction.

Set the strict dtm enforcement Parameter

The system administrator can enforce or relax the requirement to have distributed transactions commit or roll back as a logical unit by setting the **strict dtm enforcement** configuration parameter.

Note: You can also override the value of **strict dtm enforcement** using the session level **set** command with the **strict_dtm_enforcement** option.

strict dtm enforcement determines whether or not SAP ASE transaction coordination services will strictly enforce the ACID properties of distributed transactions.

Setting **strict dtm enforcement** to 1 (on) ensures that transactions are propagated only to servers that can participate in SAP ASE-coordinated transactions. If a transaction attempts to update data in a server that does not support transaction coordination services, SAP ASE aborts the transaction.

In heterogeneous environments, you may want to make use of servers that do not support transaction coordination. This includes older versions of SAP ASE and non-SAP database stores configured using CIS. Under these circumstances, you can set **strict dtm enforcement** to 0 (off). This allows SAP ASE to propagate transactions to legacy SAP ASEs and other data stores, but does not ensure that the remote work of these servers is rolled back or committed with the original transaction.

Monitoring Coordinated Transactions and Participants

SAP ASE tracks information about the status of work done in subordinate servers using data in the `sybsystemdbdbo.syscoordinations` system table.

The **sp_transactions** procedure also displays some data from the `syscoordinations` table for in-progress, remote transactions.

See the *Reference Manual: Tables*.

DTM Administration and Troubleshooting

DTM administration includes using transactions and threads of control, querying `systransactions`, executing external transactions. Troubleshooting DTM includes recovering from a system failure, heuristically completing transactions, and using configuration options to track transaction issues.

Transactions and Threads of Control

SAP ASE provides native support for the "suspend" and "join" semantics used by X/Open XA-compliant transaction managers such as Encina and TUXEDO. Transactions may be shared among different threads of execution, or may have no associated thread at all.

When a transaction has no thread associated with it, it is said to be "detached." Detached transactions are assigned a `spid` value 0. You can see the transaction `spid` value in the new `master.dbo.systransactions` table, or in output from the new **sp_transactions** procedure.

Detached transactions are meant to persist in SAP ASE, since the client application may want to reattach the original thread, or attach a new thread to the transaction. The system administrator can no longer roll back a transaction by killing its associated `spid`, as a thread is not attached to the transaction.

Transactions in a detached state may also prevent the log from being truncated with the **dump transaction** command. In extreme circumstances, detached transactions can be rolled back by using the new **dbcc complete_xact** command to heuristically complete a transaction.

The system administrator can also specify a server-wide interval after which SAP ASE automatically rolls back transactions that are in the detached state. **dtm detach timeout period** sets the amount of time, in minutes, that a distributed transaction branch can remain in the detached state. After this time has passed, SAP ASE rolls back the detached transaction.

For example, to automatically rollback detached transactions after 30 minutes, use:

```
sp_configure 'dtm detach timeout period', 30
```

Lock Manager Support for Detached Transactions

Transactions may be detached from their original threads, and have no associated `spid`. Moreover, multiple threads with different `spid` values must be able to share the same transaction locks to perform the work of a distributed transaction.

The SAP ASE lock manager uses a unique lock owner ID, rather than a `spid`, to identify transaction locks. The lock owner ID is independent from the `spid` that created the transaction, and it persists even when the transaction is detached from a thread. Lock owner IDs provide a way to support transactional locks when transactions have no associated threads, or when a new thread is attached to the transaction.

The lock owner ID is stored in the `loid` column of `syslocks`. You can determine the `loid` value of a transaction by examining **sp_lock** or **sp_transactions** output.

Examining the `spid` and `loid` columns from **sp_transactions** output provides information about a transaction and its thread of control. A `spid` value of zero indicates that the transaction is detached from its thread of control. Non-zero `spid` values indicate that the thread of control is currently attached to the transaction.

If the `loid` value in **sp_transactions** output is even, then a local transaction owns the lock. Odd `loid` values indicate that an external transaction owns the lock.

Getting Information About Distributed Transactions

`systransactions` stores information about all server transactions, and identifies each transaction and maintains information about the state of the transaction and its associated threads.

sp_transactions translates information from the `systransactions` and `syscoordinations` tables to display status conditions for active transactions.

Transaction Identification in systransactions

SAP ASE stores transaction names in a column of `varchar(255)` to accommodate the length and format of transaction names supplied by different distributed transaction protocols.

In the X/Open XA protocol, for instance, distributed transactions are assigned a transaction name consisting of both a global transaction ID (`gtrid`) and a branch qualifier. Within SAP ASE, this information is combined in the `xactname` column of the `systransactions` table.

`systransactions.xactname` stores the names of both externally-created distributed transactions (defined by an X/Open XA transaction manager or MSDTC) and local server transactions. Clients can assign any name to local transactions, within the confines of the `varchar(255)` column. Similarly, external transaction managers can use a variety of different formats to name a distributed transaction.

Transaction Keys

The transaction key, stored in `systransactions.xactkey`, acts as a unique internal handle to a server transaction.

For local transactions, `xactkey` ensures that transactions can be distinguished from one another, even if the transaction name is not unique to the server.

All system tables refer to `systransactions.xactkey` to uniquely identify a transaction. The `sysprocesses` and `syslogshold` tables are the only exceptions to this rule—they reference `systransactions.xactname` and truncate the value to a length of `varchar(64)` (for `sysprocesses`) and `varchar(67)` (for `syslogshold`), to maintain backward compatibility with earlier SAP ASE versions.

Viewing active transactions with sp_transactions

The **sp_transactions** procedure translates information from `systransactions` and `syscoordinations` to provide information about active transactions.

When used without keywords, **sp_transactions** displays information about all active transactions:

```
sp_transactions
```

```
xactkey                          type        coordinator starttime
 state             connection dbid   spid   loid
 failover                   srvname                          namelen
 xactname
 ------------------------------ ----------- -----------
 --------------------
 ------------------ ---------- ------ ------ -----------
 -------------------------- ------------------------------ -------
 ------------------------------------------------------------------
 0x00000b1700040000dd6821390001 Local       None        Jun 1 1999
3:47PM
 Begun             Attached        1   1      2
 Resident Tx               NULL                                  17
 $user_transaction
 0x00000b1700040000dd6821390001 Remote      ASTC        Jun 1 1999
3:47PM
 Begun             NA              0   8      0
 Resident Tx               caserv2                          108

 00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002
```

Identify Local, Remote, and External Transactions

The `type` column of `systransactions` indicates whether the transaction is local, remote, or external.

Local transactions execute on the local server (the server on which you ran **sp_transactions**). Local transactions have a null value in the “srvname” column, since the transaction takes place on the current server.

For remote transactions, **sp_transactions** lists the name of the server executing the transaction under the “srvname” column. The **sp_transactions** output above shows a remote transaction executing on the server named caserv2.

External transactions indicate that the transaction is coordinated by an external transaction coordinator, such as CICS, Encina, or the “ASTC HANDLER” process of another SAP ASE. External transactions also have a null value in the “srvname” column.

Identify the Transaction Coordinator

The `coordinator` column of `systransactions` indicates the method or protocol used to manage a transaction.

In the previous output, the local transaction **$user_transaction** does not have an external coordinator. The remote transaction taking place on caserv2 has the coordinator value "ASTC". This indicates that the transaction is coordinated using native SAP ASE coordination services.

See **sp_transactions** in the *Reference Manual* for a complete list and description of possible coordinator values.

View the Transaction Thread of Control

The `spid` column of `systransactions` displays the Process ID of the process attached to the transaction (or 0 if the transaction is detached from its thread of control).

For local transactions, the `spid` value indicates a Process ID running on the local server. For remote transactions, the `spid` indicates the Process ID of a task running on the indicated remote server. The output above shows a `spid` value of 8 running on the remote server, caserv2.

Understanding Transaction State Information

The `state` column of `systransactions` displays information about the current state of each transaction.

At any given time, a local or external transaction may be executing a command, aborted, committed, and so forth. Additionally, distributed transactions can be in a prepared state, or can be heuristically completed or rolled back.

The `connection` column displays information about the state of the transaction's connection. You can use this information to determine whether a transaction is currently attached to or detached from a process. Transactions in X/Open XA environments may become detached from their initiating process, in response to requests from the transaction manager.

See **sp_transactions** in the *Reference Manual: Procedures* for a complete list and description of possible coordinator values.

Limiting sp_transactions Output to Specific States

Use **sp_transactions** with the **state** keyword to limit output to the specified transaction state. For example, this displays information only for distributed transactions that have been prepared:

```
sp_transactions "state", "Prepared"
```

Transaction Failover Information

The `failover` column displays special information for servers operating in high availability environments.

In high availability environments, prepared transactions may be transferred to a secondary companion server if the original server experiences a critical failure. The `failover` column can display three possible failover states that indicate how and where the transaction is executing:

- "Resident Tx" – appears under normal operating conditions, and on systems that do not utilize SAP ASE high availability features. "Resident Tx" means that the transaction was started and is executing on a primary SAP ASE.
- Failed-over Tx" – appears after there has been a failover to a secondary companion server. "Failed-over Tx" means that a transaction originally started on a primary server and reached the prepared state, but was automatically migrated to the secondary companion server (for example, as a result of a system failure on the primary server). The migration of a prepared transaction occurs transparently to an external coordinating service.
- Tx by Failover-Conn" – appears after there has been a failover to a secondary companion server. "Tx by Failover-Conn" indicates that the application or client attempted to start the transaction on a primary server, but the primary server was not available due to a connection failover. When this occurs, the transaction is automatically started on the secondary companion server, and the transaction is marked "Tx by Failover-Conn".

Determining the Commit Node and gtrid with sp_transactions

Using **sp_transactions** with the **xid** keyword displays the commit node, parent node, and `gtrid` of a particular transaction, and all active transactions.

However, **sp_transactions ... xid** requires that you specify a particular transaction name.

For example:

```
sp_transactions "xid", "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002"
```

```
xactkey                          type        coordinator starttime
 state             connection dbid   spid   loid
 failover                   srvname                        namelen
 xactname
 commit_node parent_node
 gtrid
 ------------------------------ ----------- -----------
 --------------------
 ----------------- ---------- ------ ------ -----------
 ---------------------------- ------------------------------ -------
 ------------------------------------------------------------------
 -------------
 -------------
 -------------
 0x00000b2500080000dd6821960001 External    ASTC        Jun 1 1999
```

```
3:47PM
 Begun             Attached        1      8         139
 Resident Tx               NULL                              108

 00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002

 caserv1 caserv1
 00000b1700040000dd6821390001-aa01f04ebb9a
```

Commit and Parent Nodes

For distributed transactions coordinated by SAP ASE, the `commit node` column lists the name of the server that executes the topmost branch of the distributed transaction. This transaction determines the commit or rollback status for all branches of the transaction.

The `parent node` column lists the name of the server that initiated the transaction. In the **sp_transactions** output above, the "commit node" and "parent node" columns list the same server, caserv1. This indicates that the distributed transaction originated on caserv1, and caserv1 propagated a branch of the transaction to the current server.

Global Transaction ID

The `gtrid` column displays the global transaction ID for distributed transactions coordinated by SAP ASE.

Transaction branches that are part of the same distributed transaction share the same `gtrid`. You can use a specific `gtrid` with the **sp_transactions** `gtrid` keyword to determine the state of other transaction branches running on the current server. This is useful for system administrators who must determine whether a particular branch of a distributed transaction should be heuristically committed or rolled back.

Note: For transactions coordinated by an X/Open XA-compliant transaction manager, MSDTC, or SYB2PC, the `gtrid` column shows the full transaction name supplied by the external coordinator.

Executing External Transactions

The transaction manager executes external transactions.

The transaction manager:

1. Initiates a **begin transaction**.
2. Initiates an **attach transaction**.

   Note: The transaction manager might perform steps 1 and 2 together.
3. The application executes DML commands.
4. Initiates a **detach transaction**.
5. Repeat steps 2 through 4, if necessary.

6. Initiates a **prepare transaction**, if the transaction is not rolled back.
7. Initiates a **commit transaction** or a **rollback transaction**.

Executing step 3 can cause the distributed transaction to roll back.

Because it is cumbersome to check the global variable before issuing every command, many user applications do not check it at all. Before version 15.0.3, if the distributed transaction rolled back, SAP ASE allowed the user application to continue issuing SQL commands. These commands executed outside the distributed transaction as independent transactions. A SQL command that should have been included in a rollback transaction could be committed independently of that transaction, causing transactionally inconsistent data.

In versions 15.0.3 and later, SAP ASE automatically prevents SQL commands that are intended to execute inside a distributed transaction from executing outside it. The user application no longer has to check the global variable before every command; when a transaction is implicitly aborted, an error message (3953) appears, saying "Cannot execute the command because the external transaction has been rolled back." This message disappears when a **detach transaction** command is issued.

To suppress the 3953 error messages and let SAP ASE restore the former behavior, executing SQL commands even if the DTM transaction is not active, start SAP ASE using trace flag -T3955.

Crash Recovery Procedures for Distributed Transactions

During crash recovery, SAP ASE must resolve distributed transactions that it discovers in the prepared state.

The method used to resolve prepared transactions depends on the coordination method or coordination protocol used to manage the distributed transaction.

Note: Crash recovery procedures for distributed transaction are not performed during normal database recovery for **load database** or **load transaction** commands. If **load database** or **load transaction** applies any transactions that are in-doubt, SAP ASE aborts those transactions before bringing the associated database online.

Transactions Coordinated with MSDTC During Crash Recovery

Prepared transactions that were coordinating using MSDTC are rolled forward or backward depending on the commit status of the master transaction.

During recovery, SAP ASE initiates contact with MSDTC to determine the commit status of the master transaction, and commits or rolls back the prepared transaction accordingly. If it cannot contact MSDTC, the recovery procedure waits until contact is established. Further recovery does not take place until SAP ASE has established contact with MSDTC.

Transactions Coordinated by SAP ASE or X/Open XA During Crash Recovery

During crash recovery, SAP ASE may also encounter prepared transactions that were coordinated using SAP ASE transaction coordination services or the X/Open XA protocol.

Upon encountering these transactions, the local server must wait for the coordinating SAP ASE or the external transaction coordinator to initiate contact and indicate whether the prepared transaction should commit or roll back.

To speed the recovery process, SAP ASE restores each of these transactions to their condition prior to the failure. The transaction manager creates a new transaction with the original transaction ID, and the lock manager applies locks to protect data that the original transaction was modifying. The restored transaction remains in a prepared state but is detached, having no thread associated with it.

Once the transaction's coordinator contacts SAP ASE, the transaction manager can commit or roll back the transaction.

Using this recovery mechanism, the server can bring a database online even when the coordinating SAP ASE or external transaction manager has not yet attempted to resolve the prepared transaction. Other clients and transactions can resume work on the local data, since the prepared transaction holds the locks it did prior to recovery. The prepared transaction itself is ready to commit or roll back once contacted by its coordinator.

When the controlling SAP ASE or external transaction manager cannot complete the transaction, the system administrator can heuristically complete the transaction to free its locks and transaction resources.

See also

- *Heuristically Completing Transactions* on page 176

Transactions Coordinated with SYB2PC During Crash Recovery

Prepared transactions that were coordinated using the SYB2PC protocol are rolled forward or backward depending on the commit status of the master transaction.

During recovery, SAP ASE initiates contact with the commit service to determine the commit status of the master transaction, and commits or rolls back the prepared transaction accordingly. If it cannot contact the commit service, SAP ASE does not bring the database online. However, SAP ASE does proceed to recover other databases in the system.

This recovery method was used for SYB2PC transactions in earlier SAP ASE versions and is unchanged with SAP ASE version 12.5 and later.

Heuristically Completing Transactions

SAP ASE includes the **dbcc complete_xact** command to facilitate heuristic completion of transactions.

dbcc complete_xact resolves a transaction by either committing or rolling back its work, freeing whatever resources the transaction was using. The command is provided for those

cases where only the system administrator can properly resolve a prepared transaction, or for when the system administrator must resolve a transaction without waiting for the transaction's coordinator.

For example, in , heuristic completion may be considered if all remote SAP ASEs have prepared their transactions, but the network connection to ASE1 was *permanently lost*. The remote SAP ASEs will maintain their transactions in a prepared state until contacted by the coordination service from ASE1. In this case, only the system administrator for ASE2, ASE3, and ASE4 can properly resolve the prepared transactions. Heuristically completing the prepared transaction in ASE3 frees up transaction and lock resources, and records the commit status in `systransactions` for later use by the transaction coordinator. Heuristically completing the transaction in ASE2 also completes the transaction propagated to ASE4.

See also

• *Transactions Coordinated by SAP ASE or X/Open XA During Crash Recovery* on page 176

Completing Prepared Transactions

By using **dbcc complete_xact**, the system administrator forces SAP ASE to commit or roll back a branch of a distributed transaction.

After heuristically completing a prepared transaction, SAP ASE records the transaction's commit status in `master.dbo.systransactions` so that the transaction's coordinator —SAP ASE, MSDTC, or an X/Open XA transaction manager—can know whether the transaction was committed or rolled back.

Warning! Heuristically completing a prepared transaction can cause inconsistent results for an entire distributed transaction. The system administrator's decision to heuristically commit or roll back a transaction may contradict the decision made by the coordinating SAP ASE or transaction protocol.

Before heuristically completing a transaction, the system administrator should make every effort to determine whether the coordinating SAP ASE or transaction protocol decided to commit or roll back the distributed transaction.

SAP ASE propagates the command to heuristically commit or abort a transaction to any participant servers that it coordinated for the transaction branch. For example, if in you heuristically commit the transaction on ASE2, ASE2 propagates the command to ASE4 so that the transaction on ASE4 also commits.

dbcc complete_xact requires that you supply an active transaction name and desired outcome for the transaction.

For example, the following command heuristically commits a transaction:

```
dbcc complete_xact "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002", "commit"
```

Maintaining a Transaction's Commit Status

When the system administrator heuristically completes a prepared transaction, SAP ASE maintains information about the transaction's commit status in `master.dbo.systransactions`. This information is maintained so external transaction coordinators can detect the presence of heuristically completed transactions.

If the external coordinator is another SAP ASE, the server examines the commit status and logs a warning message if the heuristic completion conflicts with the commit status of the distributed transaction. After examining the commit status, the coordinating SAP ASE clears the commit status information from `systransactions`.

If the external coordinator is an X/Open XA-compliant transaction manager, the transaction manager does not log warning message when the heuristic completion conflicts with the distributed transaction. However, X/Open XA-compliant transaction managers clear the commit status information from `systransactions`.

Manually Clearing the Commit Status

dbcc forget_xact purges the commit status of a heuristically completed transaction from `systransactions`.

dbcc forget_xact can be used when the system administrator does not want the coordinating service to have knowledge that a transaction was heuristically completed, or when an external coordinator will not be available to clear information from `systransactions`.

See **dbcc** in the *Reference Manual: Commands*.

Completing Transactions That Are Not Prepared

dbcc complete_xact can be used to roll back SAP ASE-coordinated transactions that have not yet reached the prepared state.

Heuristically rolling back a transaction that has not yet been prepared does not pose a risk to the distributed transaction, since the coordinating server can recognize that the transaction failed to prepare its work. Under these circumstances, the coordinating SAP ASE can roll back the entire distributed transaction to preserve consistency.

When you heuristically roll back an SAP ASE transaction that has not yet been prepared, SAP ASE *does not* record the heuristic roll back in `systransactions`. Instead, an informational message is printed to the screen and recorded in the server's error log.

Determining the Commit Status for SAP ASE Transactions

If the distributed transaction branch you want to commit or roll back is coordinated by SAP ASE, you can use **sp_transactions** to determine the commit status of the distributed transaction.

Note: These steps cannot be used with distributed transactions that are coordinated by the X/Open XA protocol, MSDTC, or SYB2PC.

1. In the server that is executing the transaction branch you want to complete, use **sp_transactions** with the **xid** keyword to display information about the transaction. Record the commit node and `gtrid` of the transaction. For example:

```
sp_transactions "xid", "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002"
```

```
xactkey                          type        coordinator starttime
 state             connection dbid   spid   loid
 failover                  srvname                         namelen
 xactname
 commit_node parent_node
 gtrid
 ------------------------------ ----------- -----------
 --------------------
 ----------------- ---------- ------ ------ -----------
 -------------------------- ------------------------------
-------

-----------------------------------------------------------------
 -------------
 -------------
 -------------
 0x00000b2500080000dd6821960001 External    ASTC       Jun 1 1999
3:47PM
 Begun             Attached        1      8         139
 Resident Tx               NULL                              108

 00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002

 caserv1 sfserv
 00000b1700040000dd6821390001-aa01f04ebb9a
```

   In this example, the commit node for the distributed transaction is “caserv1” and the `gtrid` is “00000b1700040000dd6821390001-aa01f04ebb9a”.

2. Log on to the server indicated by the commit node. For example:

```
isql -Usa -Psa_password -Scaserv1
```

3. Use **sp_transactions** with the `gtrid` keyword to determine the commit status of the distributed transaction having the `gtrid` obtained in step 1:

```
sp_transactions "gtrid", "00000b1700040000dd6821390001-
aa01f04ebb9a"
```

```
xactkey type coordinator starttime
state connection dbid   spid   loid
failover srvname namelen
xactname
commit_node
parent_node
------------------------------ ----------- -----------
--------------------
```

```
---------------- ---------- ------ ------ -----------
------------------------- ------------------------------ -------

------------------------------------------------------------------
 -------------
 -------------
 0x00000b1700040000dd6821390001 Local       None        Jun 1 1999
3:47PM
 Committed Attached        1      1           2
 Resident Tx                NULL                                17
 $user_transaction

 caserv1
 caserv1
```

In this example, the local transaction with the specified `gtrid` has committed, as indicated by the "state" column. The system administrator should heuristically *commit* the prepared transaction examined in step 1.

4. Using an account with system administrator privileges, log on to the server that is executing the transaction branch you want to complete:

```
isql -Usa -Psa_password -Ssfserv
```

5. Use **dbcc complete_xact** to commit the transaction. In this example, the system administrator should use the **commit** keyword to maintain consistency with the distributed transaction:

```
dbcc complete_xact "00000b1700040000dd6821390001-
aa01f04ebb9a-00000b1700040000dd6821390001-aa01f04ebb9a-caserv1-
caserv1-0002", "commit"
```

Troubleshooting for Transactions Coordinated by External Transaction Managers

If a transaction is coordinated by an external transaction manager using X/Open XA protocol or through Adpative Server transaction coodination services of another SAP ASE, then DDL commands are not allowed within the transaction. This behavior applies even if the database option **ddl in tran** is enabled.

SAP ASE Implicit Rollback in External Transactions

If you encounter errors in an external transaction (for example, deadlocks, aborted update triggers, and so on) , SAP ASE may abort the external transaction.

Although SAP ASE sends error messages for failures, applications do not always check for messages, particularily for simple inserts (for example, they may not be aware of triggers added by DBAs). It may not always be obvious from the error messages that the XA transaction has ended.

If SAP ASE aborts an external transaction and throws a SQLException, you can issue `select @@trancount`. If the value for *@@trancount* is zero, the DTM transaction was aborted.

The application should call the transaction manger (typically an application server) notifying it that the transaction aborted. If you ignore error messages, subsequent updates could take place outside the DTM transaction context (for example, local transactions). You can log the error messages and check the *@@transtate* or *@@trancount* to verify the updates occured.

The following describes a trigger that causes SAP ASE to rollback an external transaction. The insert statement contains the trigger that can potentially fail. If SAP ASE cannot issue the **insert**, the update runs the **ut.commit** function

This example (in pseudo code) assumes you are running a JTA/XA UserTransaction:

```
try {

insert into table values (xx.... )
update table
ut.commit();

} catch (SQLException sqe) {

if this is a known error then process
else
select @@trancount into count
if count == 0
then ut.rollback() }
```

If you do not include the rollback funtion, then additional updates take place outsie the JTA/XA transaction.

CHAPTER 9 Support for OData

SAP ASE support supports SAP Sybase OData Server. OData (Open Data Protocol) enables data services over RESTful HTTP, allowing you to perform operations through URIs (Universal Resource Identifiers) to access and modify information.

This section assumes you are familiar with OData protocol concepts. See the *OData Protocol Web site* for information about OData.

*OData Support in **dbsvc** Utility*

The **dbsrv** utility on Windows can now create services for OData. To create such a service, use **dbsrv -t OData**. See *Service utility (dbsrv) for Windows* in *SQL Anywhere Server Database Administration* for detailed information on **dbsrv**.

OData Server Architecture

The SAP Sybase OData Server consists of the OData Producer and an HTTP server.

The OData Server consists of:

- OData Producer – a Java servlet that uses the JDBC API to connect to an SAP ASE. The OData Producer processes OData requests and responses, and interfaces with the database. The OData Producer maps OData concepts to relational database concepts in this way:

OData Concept	Database Equivalent
Entity type	Table or view
Entity type instance	Row
Key	Primary key
Link	Foreign key
Property	Column

- An HTTP server that handles OData requests from Web clients – the OData server uses the Jetty WebServer as its HTTP server. This embedded HTTP server also acts as a Java servlet container, which is required to host the OData Producer.

  Instead of using the embedded HTTP server, you can use your own HTTP server to handle OData requests, as long as your solution can also host Java servlets. For example, you can set up an IIS or Apache server to forward requests to a Tomcat server.

OData client requests are sent to an HTTP server through URIs, and are processed by the OData Producer, which then interfaces with the database server to issue database requests and retrieve content for the OData responses.

The OData schema for each client is based on the client's database connection permissions. Clients cannot view or modify database objects for which they do not have view permission.

You can grant client access to the database using a preconfigured connection string or basic HTTP authentication.

OData Server Limitations

OData Server complies with OData protocol version 2 specifications, but has several limitations that are not explicitly defined by OData protocol definitions.

- Schema changes – restart the OData Server utility when you make changes to the database schema so that the changes can take effect and become visible to OData clients.
- **orderby** queries – sort only by entity properties. Ordering by direction is supported, but sorting by expressions is not.

Unsupported OData Protocol Features

There are several OData protocol features that are unsupported by OData Producer.

- Deep inserts and updates
- Service operations
- Dynamic properties
- Complex types
- Media types
- HTTP Etags

Security Considerations for OData Server

Take security measures into consideration before setting up OData Server.

Consideration	Description
HTTPS certification	Any HTTPS certificate details specified in the OData Server configuration file apply only to the embedded HTTP server. For more information about how HTTPS certification is handled through an alternative HTTP server, see the HTTP server documentation.

Consideration	Description
Traffic without use of SSL protocol	SAP recommends that you always use SSL in production deployments. All traffic between the OData Producer and clients is transmitted in plain text, including user IDs and passwords, when the SSLKeyStore option is not specified in the OData Server configuration file. This option is not specified in default configurations.

Configuring OData Server

Before you start OData Server, specify embedded HTTP server options, OData producer options, and database connection parameter settings.

1. In a text editor, create and open a configuration file called `server.properties`.

   Sample configuration files are located:
   - `$SYBASE/ODATA-16_0/samples/java` (`%SYBASE%\ODATA-16_0\samples\java` in Windows)
   - `$SYBASE/ODATA-16_0/samples/dotnet` (`%SYBASE%\ODATA-16_0\samples\dotnet` in Windows)
2. In the file, specify the options for the embedded HTTP server:

Option	Description
LogFile = *path-and-file-ename*	Specifies the path and file name to which the embedded HTTP server logs OData Producer output. The default behavior is to disable logging. The path is relative to the location of the server executable.
LogVerbosity = { 1 | 2 | 3 | 4}	Higher verbosity levels log additional information and include the information provided by all lower levels. Verbosity level: • 1 – returns information about any unexpected errors. • 2 – returns general information and configuration messages. • 3 – returns detailed information about HTTP requests. • 4 – returns debugging messages.
ServerPort = *port-number*	Specifies the port number on which the embedded HTTP server listens. The default setting is 80.
ShutdownListenerPort = *port_number*	Specifies the port number on which the embedded server listens for shutdown requests. The default setting is 2449.

Option	Description
SSLKeyStore = *path-and-filename*	Specifies the path and file name to a Java keystore containing an SSL certificate that the embedded HTTP server uses to encrypt traffic. SSL is enabled and unencrypted HTTP traffic is blocked when you specify option. The path is relative to the location of the server executable.
SSLKeyStorePassword = *SSLKeyStore-password*	Specifies the password that the embedded HTTP server uses to authenticate against the Java keystore identified by the SSLKeyStore option.

3. In the file, specify the OData Producer options:

Option	Description
Authentication = {none | database)	Specifies the credentials used to connect to the database. Valid options are: • (Default) **database** – indicates that users connect with personalized credentials that are appended to the **DbConnectionString** option to form their own complete database connection string. These credentials are requested using basic HTTP authentication. • **none** – indicates that all users connect using the same connection string, as indicated by the **DbConnectionString** option.
ConnectionPool-Maximum = *num-max-connections*	Indicates the maximum number of simultaneous connections that the OData Producer keeps open for use in the connection pool. The connection pool may use fewer connections depending on the server load. By default, the connection pool size is limited by the number of maximum number of simultaneous connections permitted by the database server.
DbConnection-String = *connection-string*	Specifies the connection string used to connect to the database. The connecting string should exclude the **user** and **password** parameters when the **Authentication** option is set to **database**.
DbProduct = ase	Indicates the type of database server to which the OData Producer connects.
PageSize = *num-max-entities*	Specifies the maximum number of entities to include in a retrieve entity set response before issuing a next link. The default setting is 100.
Model = *path-and-filename*	Specifies the path and file name to the OData Producer service model that indicates which tables and views are exposed in the OData metadata. The default behavior is to expose tables and views based on user privileges. Tables and views without primary keys are not exposed. The path is relative to the location of the server executable.

Option	Description
ModelConnection-String = ***connection-string***	Specifies a connection string that the OData Producer uses to validate the OSDL file during start-up. OSDL validation ensures that enlisted tables and columns exist, that key lists are used appropriately, and that the file is semantically correct. The connection string should include the **user** and **password** parameters. The default behavior is to assume that the OSDL file is valid.
ReadOnly = {true | false}	Indicates whether modification requests should be ignored. The default setting is **false**.
ServiceRoot = ***url-prefix***	Specifies the URL prefix to append to links in OData responses. This setting is required when working with certain proxy configurations, such as reverse proxies. The default behavior is to automatically determine the URL prefix, using the data stored in the request.

4. Specify the database connection parameter settings in the configuration file:

   ```
   DbConnectionString = connection-string
   ```

   This specifies the connection string used to connect to the database.

   Do not include **user** or **password** when you set **DbAuthentication** to **database**.

The configuration file should look similar to:

```
# Embedded HTTP server options
# ----------------------------
ServerPort = 8000
ShutdownListenerPort = 8083
SSLKeyStore = ../../samplekeystore.jks
SSLKeyStorePassword = password
LogFile = ../../odata.log
LogVerbosity = 1

# OData Producer options
# ----------------------
DbAuthentication = none
DbProduct = ASE
MaximumFeedSize = 100
Model = ../../model.osdl
ModelConnectionString = servername:portnumber/dbname?
user=yourusername&password=yourpassword
ReadOnly = false
ServiceRoot = localhost:8000/odata/

# Database connection parameters
# ------------------------------
DbConnectionString = servername:portnumber/dbname?
user=yourusername&password=yourpassword
```

Next

After you save the `server.properties` file, run **dbosrv16** to start OData Server with the options you specified in the configuration file:

```
dbodata server.properties
```

Setting Up an HTTP Server for OData

You can set up either the embedded HTTP server or your own HTTP server for OData operations.

Take security measures into consideration when you set up an HTTP server.

Decide whether you are setting up an embedded HTTP Server or an alternate HTTP server. If you choose:

Server Type	Steps
Embedded HTTP server	The SAP Sybase OData Server utility initiates an instance of the embedded HTTP server and automatically loads the OData Producer as a Java servlet. You cannot manually configure the embedded HTTP server, or use it to serve other non-OData content, such as HTML files. However, you can specify some HTTP server options in the OData Server configuration file. To set up and launch the embedded HTTP server: **1.** Create an OData Server configuration file that contains the settings for both the OData Producer and the embedded HTTP server. **2.** Store the configuration file on the computer that acts as the HTTP server, then load the configuration file when you run the OData Server utility at a command prompt.

Server Type	Steps
Alternate HTTP server	To use the OData Producer with an alternative HTTP server, you must deploy the OData Producer to the server. You must run the OData Producer as a Java servlet that can be loaded into an HTTP server. For example, you can use Tomcat as a Java servlet container and pair it with an Apache HTTP server. **Note:** IIS (Internet Information Services, the Microsoft Web server) cannot execute Java servlets, but you can configure a connector that redirects servlet requests from an IIS server to another server that is able to run them. The process of deploying the OData Producer differs depending on your HTTP server. For more information, see your HTTP server documentation. **1.** Create an OData Server configuration file. Any configuration options that are specific to the embedded HTTP server, including security considerations, are not applied to the alternative HTTP server. The OData Producer respects the logging configuration of the HTTP server. For more information about HTTP server logging, see your HTTP server documentation. **2.** Copy the following the OData Server configuration file (`server.properties`) to the `lib` directory of your Web application server, along with the following files from your SAP ASE installation. On UNIX: • `$SYBASE/jConnect-7_0/classes/jconn.jar` • `$SYBASE/ODATA-16_0/classes/dbodata.jar` On Windows: • `%SYBASE%\jConnect-7_0\classes\jconn.jar` • `%SYBASE%\ODATA-16_0\classes\dbodata.jar` **3.** Load the SAPSybaseODataServlet class, which implements the J2EE.

Create an OData Producer Service Model

Use the **Model** option in the OData Server configuration file to create an OData Producer service model.

You can create an OData Producer service model to expose specific tables and views by defining a namespace (the model) in a text file that complies to the OData Service Definition Language (OSDL) syntax. The model is applied to the OData Producer when you reference the text file using the **Model** option in your OData Server configuration file.

Syntax

```
service [namespace "namespace-name"] {
    "owner"."{table-name | view-name}" [ keys("column-name", ...) ]
    ...
  }
```

Parameters

- ***namespace-name* –** is the name of the service you define. Append *_Container* to *service-name* to generate a container name.
- ***owner* –** is the name of the owner of the table or view.
- ***table-name* | *view-name* –** is the name of the table or view to expose in the OData metadata.
- ***column-name* –** is a column name to use as a primary key when one is not specified in the table or view defined by *table-name* or *view-name*. You can reference multiple column names.

Examples

- **Exposing a table and a view –** this service model exposes a table and a view:

```
service namespace "DBServer" {
    "dba"."TableWithPrimaryKey";
    "dba"."ViewWithoutPrimaryKey" keys("primary_key1",
"primary_key2");
}
```

Usage

- You can define a service that contains references to multiple tables or views.
- When using the **keys** parameter:
  - Specify a key list when referencing a table or view that does not contain a primary key.
  - Do not specify a key list when referencing a table that contains a primary key.

The model is applied to the OData Producer when you reference the text file using the **Model** option in your OData Server configuration file.

OData Server Sample Files

SAP ASE includes OData Server sample files to illustrate how to run OData Server, set up an OData client, and send requests and responses between them.

There are two sample files you can run, both located in `$ASE/ASE/`(`%ASE%\ASE\` in Windows):

- The OData SalesOrder sample – illustrates how to use a Microsoft .NET OData Client to send OData requests to an OData Server that connects to a SAP ASE sample database. See `$ASE/ASE/ODataSalesOrders/readme.txt` (`%ASE%\ASE\ODataSalesOrders\readme.txt` in Windows).
- The OData Security sample – illustrates how to use an OData4J Java client to send OData requests over HTTPS to an OData Server that connects to an SAP ASE sample database. This shows you how an OData Producer service model works with database

authentication. See `$ASE/ASE/ODataSecurity/readme.txt` (`%ASE%\ASE\ODataSecurity\readme.txt` in Windows).

Starting and Stopping OData Server

Use the **dbosrv16** utility to start, and **dbostop** to stop the embedded HTTP server and the OData Producer.

The SAP ASE installer creates all the folders and copies the files necessary to run OData. After you successfully install SAP ASE, start or stop the embedded HTTP server and the OData Producer by using:

Command	Description
dbosrv16	The utility, located in `$SYBASE/ODATA-16_0/bin64` (`%SYBASE%\ODATA-16_0\bin64` for Windows), starts the embedded HTTP server and the OData Producer by invoking the configuration file you created to set up parameters and options. The syntax is: `dbosrv16 server.properties` This specifies name of the configuration file you used to set up the database connection. **dbosrv16** contains both the servlet implementation and the components necessary to launch its own HTTP server.

dbostop	The utility, located in `$SYBASE/ODATA-16_0/bin64` (`%SYBASE%\ODATA-16_0\bin64` for Windows), stops the embedded HTTP server and the OData Producer. The syntax is:
	`dbostop [-q] {-f properties-filename | -p port-number}`
	where:
	• **-f *properties-filename*** – uses the port number specified by the ShutdownListenerPort option to send a shutdown request. *properties-filename* is the file name with which you started the server. If you do not include **-f**, **dbostop** attempts to shut down the embedded HTTP server and the OData Producer on port 2449.
	• **-p *port-number*** – is the value of the **ShutdownListenerPort** option you specified in the OData Server configuration file that you used to start OData Server. Using **-p** specifies the port number to send the shutdown request to. The default value is 2449. This option is:
	  • Not required if your `server.properties` configuration file already includes the `shutdownlistener` port.
	  • Required if you do not specify the `server.properties` configuration file in the command.
	  If you do not include **-p**, **dbostop** attempts to shut down the embedded HTTP server and the OData Producer on port 2449
	• **-q** – attempts to shut a server down quietly; no message appears.

Any time you change the database schema, stop OData Server with **dbostop**, then restart it with **dbosrv16**. This allows the changes to take effect and become visible to OData clients.

CHAPTER 10 Creating and Using Segments

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

Segment names are used in **create table** and **create index** commands to place tables or indexes on specific database devices. Using segments can improve SAP ASE performance and give the system administrator or database owner increased control over the placement, size, and space usage of database objects.

Create segments within a database to describe the database devices that are allocated to the database. Each SAP ASE database can contain up to 32 segments, including the system-defined segments. Before assigning segment names, you must initialize the database devices with **disk init** and then make them available to the database with **create database** or **alter database**.

See *Performance and Tuning Series: Physical Database Tuning > Controlling Physical Data Placement* for information about how segments can improve system performance.

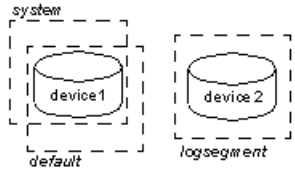
System-Defined Segments

When you create a database, SAP ASE creates three segments in the database.

- `system` – stores the database's system tables.
- `logsegment` – stores the database's transaction log.
- `default` – stores all other database objects—unless you create additional segments and store tables or indexes on the new segments by using **create table...on** *segment_name* or **create index...on** *segment_name*.

If you create a database on a single database device, the `system`, `default`, and `logsegment` segments are created on the same device. If you use the **log on** clause to place the transaction log on a separate device, the segments resemble those shown below:

Figure 13: System-Defined Segments



Although you can add and drop user-defined segments, you cannot drop the default, system, or log segments from a database. A database must have at least one of each type of system-defined segment: `system`, `logsegment`, and `default`.

These are the commands and system procedures for managing segments:

- **sp_addsegment** – defines a segment in a database.
- **create table** and **create index** – creates a database object on a segment.
- **sp_dropsegment** – removes a segment from a database or removes a single device from the scope of a segment.
- **sp_extendsegment** – adds devices to an existing segment.
- **sp_placeobject** – assigns future space allocations for a table or an index partition to a specific segmen.t
- **sp_helpsegment** – displays the segment allocation for a database or data on a particular segment.
- **sp_helpdb** – displays the segments on each database device.
- **sp_help** – displays information about a table, including the segment where the table resides.
- **sp_helpindex** – displays information about a table's indexes, including the segments where the indexes reside.

See also

- *Chapter 6, Creating and Managing User Databases* on page 117

Segment Usage in SAP ASE

When you add a new device to a database, SAP ASE places the new device in a default pool of space (the database's `default` and `system` segments).

This increases the total amount of space available to the database, but it does not determine which objects occupy that new space. Any table or index may grow to fill the entire pool of space, leaving critical tables with no room for expansion. Several heavily used tables and indexes may be placed on a single physical device in the default pool of space, resulting in poor I/O performance.

When you create an object on a segment, the object can use all the database devices that are available in the segment, but no other devices. You can use segments to control the space that is available to individual objects.

The following sections describe how to use segments to control disk space usage and to improve performance.

Controlling Space Usage

If you assign noncritical objects to a segment, those objects cannot grow beyond the space that is available in the segment's devices.

Conversely, if you assign a critical table to a segment, and the segment's devices are not available to other segments, no other objects compete with that table for space.

When the devices in a segment become full, you can extend the segment to include additional devices or device fragments as needed. Segments also allow you to use thresholds to warn you when space becomes low on a particular database segment.

If you create additional segments for data, you can create new threshold procedures for each segment.

Use Segments to Allocate Database Objects

In a large, multidatabase or multidrive SAP ASE environment, you can enhance system performance by paying careful attention to the allocation of space to databases and the placement of database objects on physical devices.

Ideally, each database has exclusive use of database devices, that is, it does not share a physical disk with another database. In most cases, you can improve performance by placing heavily used database objects on dedicated physical disks or by splitting large tables across several physical disks.

Separating Tables, Indexes, and Logs

Generally, placing a table on one physical device, its nonclustered indexes on a second physical device, and the transaction log on a third physical device improves performance.

Using separate physical devices (disk controllers) reduces the time required to read or write to the disk. If you cannot devote entire devices in this way, at least restrict all nonclustered indexes to a dedicated physical device.

The **log on** extension to **create database** (or **sp_logdevice**) places the transaction log on a separate physical disk. Use segments to place tables and indexes on specific physical devices.

Splitting Tables

To improve the overall read performance of a table, split a large, heavily used table across devices on separate disk controllers.

When a large table exists on multiple devices, it is more likely that small, simultaneous reads take place on different disks.

You can split a table across devices using these different methods, each of which requires the use of segments:

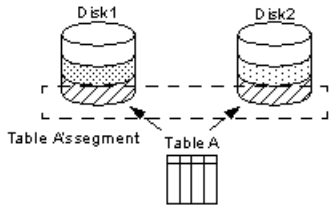
- Use table partitioning.
- If the table has a clustered index, use partial loading.
- If the table contains `text` or `image` datatypes, separate the text chain from other data.

Partitioning Tables

Partitioning a table creates multiple page chains for the table and distributes those page chains over all the devices in the table's segment.

Partitioning a table increases both insert and read performance, since multiple page chains are available for insertions.

Figure 14: Partitioning a table across physical devices



Before you can partition a table, you must create the table on a segment that contains a specified number of devices. See *Performance and Tuning Series: Physical Database Tuning > Controlling Physical Data Placement* for information about partitioning tables using **alter table**.

Partial Loading

To split a table with a clustered index, use **sp_placeobject** with multiple **load** commands to load different parts of the table onto different segments.

This method can be difficult to execute and maintain, but it does allow you to split tables and their clustered indexes across physical devices.

Separating Text and Image Columns

SAP ASE stores the data for `text` and `image` columns on a separate chain of data pages.

By default, this text chain is placed on the same segment as the table's other data. Since reading a text column requires a read operation for the text pointer in the base table and an additional read operation on the text page in the separate text chain, placing the text chain and base table data on a separate physical device can improve performance.

Moving a Table to Another Device

You can also use segments to move a table from one device to another using the **create clustered index** command.

Clustered indexes, where the bottom or leaf level of the index contains the actual data, are on the same segment as the table. Therefore, you can completely move a table by dropping its clustered index (if one exists), and creating or re-creating a clustered index on the desired segment.

Creating Segments

SAP ASE includes two methods for creating a segment in a database.

- Use **disk init** to initialize the physical device, or

- Use the **on** clause to **create database** or **alter database** to make the database device available to the database. This automatically adds the new device to the database's `default` and `system` segments.

Once the database device exists and is available to the database, use **sp_addsegment** to define the segment in the database.

See the *Reference Manual: Procedures*.

This statement creates the segment `seg_mydisk1` on the database device `mydisk1`:

```
sp_addsegment seg_mydisk1, mydata, mydisk1
```

Changing the Scope of Segments

When you use segments, you must also manage their scope—the number of database devices to which each segment points. You can extend or reduce the scope of a segment.

Extending the Scope of Segments

You may need to extend a segment if the database object or objects assigned to the segment run out of space. **sp_extendsegment** adds database devices to an existing segment.

Before you can extend a segment:

- The database device must be listed in `sysdevices`,
- The database device must be available to the database you are extending, and
- The segment name must exist in the current database.

The following example adds the database device `pubs_dev2` to an existing segment named `bigseg`:

```
sp_extendsegment bigseg, pubs2, pubs_dev2
```

To extend the `default` segment in your database, place the word "default" in quotes:

```
sp_extendsegment "default", mydata, newdevice
```

See the *Reference Manual: Procedures*.

Automatically Extending the Scope of a Segment

If you use **alter database** to add space on a database device that is new to the database, the `system` and `default` segments are extended to include the new space. Thus, the scope of the `system` and `default` segments is extended each time you add a new device to the database.

If you use **alter database** to assign additional space on an existing database device, all the segments mapped to the existing device are extended to include the new device fragment.

For example, assume that you initialized a 4MB device named `newdev`, allocated 2MB of the device to `mydata`, and assigned the 2MB to the `testseg` segment:

```
alter database mydata on newdev = "2M"

sp_addsegment testseg, mydata, newdev
```

If you later alter `mydata` to use the remaining space on `newdev`, the remaining space fragment is automatically mapped to the `testseg` segment:

```
alter database mydata on newdev = "2M"
```

Reducing the Scope of a Segment

You may need to reduce the scope of a segment if it includes database devices that you want to reserve exclusively for other segments.

For example, if you add a new database device that is to be used exclusively for one table, reduce the scope of the `default` and `system` segments so that they no longer point to the new device.

Use **sp_dropsegment** to drop a single database device from a segment, reducing the segment's scope.

sp_dropsegment drops only the given `device` from the scope of devices spanned by the segment. You can also use **sp_dropsegment** to remove an entire segment from the database.

See the *Reference Manual: Procedures*.

This example removes the database device `pubs_dev2` from the scope of `bigseg`:

```
sp_dropsegment bigseg, pubs2, pubs_dev2
```

Assigning Database Objects to Segments

Assign new or existing database objects to user-defined segments to restrict new objects to one or more devices, place a table and its index on separate devices, or split an existing object over multiple devices.

Creating New Objects on Segments

To place a new object on a segment, first create the new segment.

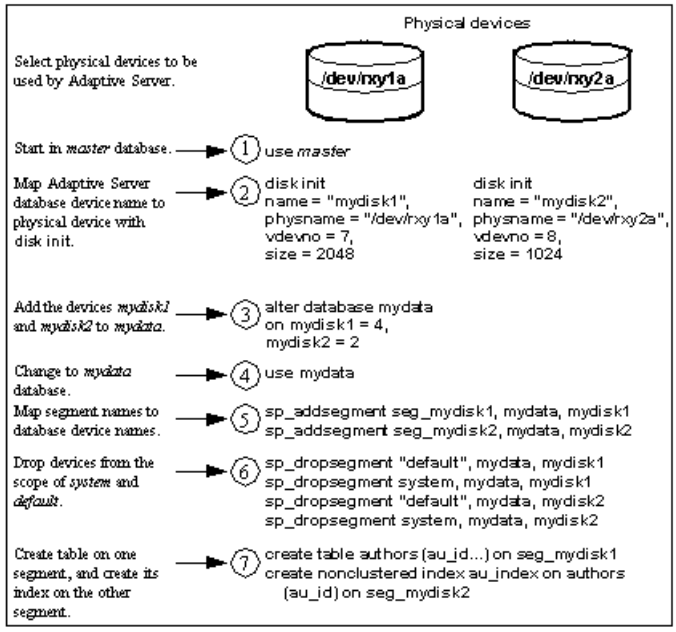
You may also want to change the scope of this segment (or other segments) so that it points only to the desired database devices. When you add a new database device to a database, it is automatically added to the scope of the `default` and `system` segments.

After you have defined the segment in the current database, use **create table** or **create index** with the optional **on** *segment_name* clause to create the object on the segment.

See the *Reference Manual: Procedures*.

This figure summarizes the sequence of Transact-SQL commands used to create tables and indexes on specific physical disks on a server using 2K logical page size.

Figure 15: Creating Objects on Specific Devices Using Segments



Placing Existing Objects on Segments

sp_placeobject does not remove an object from its allocated segment. However, it causes all further disk allocation for that object to occur on the new segment it specifies.
For example, for all further disk allocation for the `mytab` table to take place on `bigseg`, use:

```
sp_placeobject bigseg, mytab
```

sp_placeobject does not move an object from one database device to another. Any pages allocated on the first device remain allocated; any data written to the first device remains on the device. **sp_placeobject** affects only future space allocations.

After you have used **sp_placeobject**, if you then execute **dbcc checkalloc** you see this message for each object that is split across segments:

```
Extent not within segment: Object object_name, indid index_id
includes extents on allocation page page_number which is not in
segment segment_name.
```

You can ignore this message.

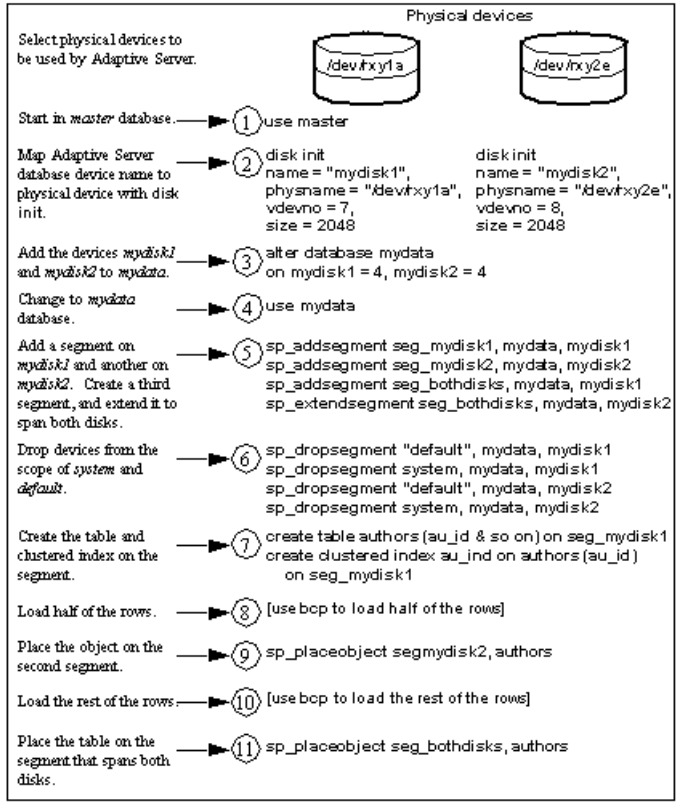
See the *Reference Manual: Procedures*.

Split large tables across segments that are located on separate disk controllers to improve performance for high-volume, multiuser applications.

The order of steps is important; in particular, create the clustered index before you place the table on the second segment.

This figure summarizes how to split a table across two segments on a server using a 2K logical page size:

Figure 16: Splitting a Large Table Across Two Segments



The balance of disk allocation may change over time if the table is updated frequently. To guarantee that the speed advantages are maintained, you may need to drop and re-create the table.

Placing Text Pages on a Separate Device

When you create a table with `text` or `image` columns, the data is stored on a separate chain of text pages.

A table with `text` or `image` columns has an additional entry in `sysindexes` for the text chain, with the name column set to the name of the table preceded by the letter “t” and an `indid` of 255.

Use **sp_placeobject** to store the text chain on a separate device, giving both the table name and the name of the text chain from `sysindexes`:

```
sp_placeobject textseg, "mytab.tmytab"
```

Note: By default, a chain of text pages is placed on the same segment as its table. After you execute **sp_placeobject**, pages that were previously written on the old device remain allocated, but all new allocations take place on the new segment.

If you want the text pages to be on a particular segment, first create the table on that segment (allocating the initial extent on that segment), then create a clustered index on the table to move the rest of the data to the segment.

Creating Clustered Indexes on Segments

The bottom, or leaf level, of a clustered index contains the data. Therefore, a table and its clustered index are on the same segment.

If you create a table on one segment and its clustered index on a different segment, the table migrates to the segment where you created the clustered index. This provides a quick and easy way to move a table to other devices in your database.

This example creates a clustered index, without specifying the segment name, using a table on the `new_space` segment :

```
create clustered index mytabl_cix
    on mytabl(c1)
```

```
sp_helpsegment new_space
```

```
segment     name           status
-------     ------------- ------
      3     new_space           0

device            size       free_pages
----------------- ------     ----------
newdevice         3.0MB            1523

total_size    total_pages    free_pages  used_pages   reserved_pages
------------- -----------    ----------- -----------   --------------
3.0MB                1536           1530           6                 0
```

If you have placed a table on a segment, and you must create a clustered index, use the **on** *segment_name* clause, or the table migrates to the `default` segment.

See **create index** in the *Reference Manual: Commands*.

Dropping Segments

When you use **sp_dropsegment** with only a segment name and the database name, the named segment is dropped from the database.

However, you cannot drop a segment as long as database objects are still assigned to it. You must first assign the objects to another segments or drop the objects, then drop the segment.

You cannot completely drop the default, system, or log segment from a database. A database must have at least one default, system, and log segment. You can, however, reduce the scope of these segments.

Note: Dropping a segment removes its name from the list of segments in the database, but it does not remove database devices from the allocation for that database, nor does it remove objects from devices. If you drop all segments from a database device, the space is still allocated to the database but cannot be used for database objects. **dbcc checkcatalog** reports "Missing segment in Sysusages segmap." To make a device available to a database, use **sp_extendsegment** to map the device to the database's default segment:

```
sp_extendsegment "default", dbname, devname
```

See the *Reference Manual: Procedures*.

Getting Information About Segments

SAP ASE includes a number of system procedures that provide information about segments.

- **sp_helpsegment** – lists the segments in a database or displays information about a particular segment in the database.
- **sp_helpdb** – displays information about the relationship between devices and segments in a database.
- **sp_help** and **sp_helpindex** – display information about tables and indexes, including the segment to which the object is assigned.

sp_helpsegment

sp_helpsegment, when used without an argument, displays information about all of the segments in the database from which you execute it.
For example:

```
sp_helpsegment
```

```
segment name                          status
------- ------------------------------ ------
      0 system                              0
      1 default                             1
      2 logsegment                          0
```

```
       3 seg1                                0
       4 seg2                                0
```

Specify the segment name as an argument for information about a particular segment. Use quotes when requesting information about the `default` segment:

```
sp_helpsegment "default"
```

This example displays information about `seg1`:

```
sp_helpsegment seg1
```

```
segment name                          status
------- ------------------------------ ------
      4 seg1                                0

device                  size             free_pages
----------------------- ---------------- -----------
user_data10             15.0MB                  6440
user_data11             15.0MB                  6440
user_data12             15.0MB                  6440

table_name              index_name            indid
----------------------- ---------------------- ------
customer                customer                   0

total_size      total_pages free_pages  used_pages
--------------- ----------- ----------- -----------
45.0MB                23040       19320        3720
```

sp_helpdb

Execute **sp_helpdb** within a database, and specify the database's name to see information about the segments in the database.
For example:

```
sp_helpdb pubs2
```

```
name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa            4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size         usage        created     free kbytes
------------------- ------------ ------------ ----------   -------
-----
master              10.0MB        data and log  Apr 13
2005          1792

pubs_2_dev        10.0MB       data and log  Apr 13 2005        9888

device                  segment
----------------------- ----------------------
master                  default
master                  logsegment
master                  system
pubs_2_dev              default
```

```
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_help and sp_helpindex

Execute **sp_help** and **sp_helpindex** in a database, and specify a table's name to see information about which segments store the table or its indexes.
For example:

```
sp_helpindex authors
```

```
index_name  index_keys  index_description  index_max_rows_per_page
     index_fillfactor  index_reservepagegap  index_created
     index_local
----------  ----------  -----------------  ----------------------
     ---------------  --------------------  -------------
     -----------
auidind     au_id       clustered,unique                          0
                    0                    0   Apr26 2005  4:04PM
     Global Index
aunwind     au_lname,au_fname  nonclustered,unique               0
                    0                    0   Apr26 2005  4:04PM
     Global Index
(2 rows affected)
index_ptn_name        index_ptn_seg
--------------------  --------------
auidind_400001425     default
aunmind_400001425     default
```

Segments and System Tables

Three system tables store information about segments: `master..sysusages` and two system tables in the user database, `sysindexes` and `syssegments`.

sp_helpsegment uses these tables and finds the database device name in `sysdevices`.

When you allocate a device to a database with **create database** or **alter database**, SAP ASE adds a row to `master..sysusages`. The `segmap` column in `sysusages` provides bitmaps to the segments in the database for each device.

create database also creates the `syssegments` table in the user database with these default entries:

```
segment name            status
------- --------------- ------
      0 system              0
      1 default             1
      2 logsegment          0
```

Executing **sp_addsegment**:

- Adds a new row to the `syssegments` table in the user database, and
- Updates the `segmap` in `master..sysusages`.

When you create a table or an index partition, SAP ASE adds a new row to `sysindexes`. The `segment` column in that table stores the segment number, showing where the server allocates new space for the object. If you do not specify a segment name when you create the object, it is placed on the `default` segment; otherwise, it is placed on the specified segment.

If you create a table containing `text` or `image` columns, a second row is also added to `sysindexes` for the linked list of text pages; by default, the chain of text pages is stored on the same segment as the table. An example using **sp_placeobject** to put the text chain on its own segment is included in the section below.

The `name` from `syssegments` is used in **create table** and **create index** statements. The `status` column indicates which segment is the default segment.

A Segment Tutorial

There are a number of things to consider when you are working with segments and devices.

- If you assign space in fragments, each fragment has an entry in `sysusages`.
- When you assign an additional fragment of a device to a database, all segments mapped to the existing fragment are mapped to the new fragment.
- If you use **alter database** to add space on a device that is new to the database, the `system` and `default` segments are automatically mapped to the new space.

This tutorial shows how to create a user segment and how to remove all other segment mappings from the device. The examples in this section assume a server using 2K logical page sizes. The tutorial begins with a new database, created with one device for the database objects and another for the transaction log:

```
create database mydata on bigdevice = "5M"
     log on logdev = "4M"
```

Now, if you **use mydata**, and run **sp_helpdb**, you see:

```
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created      status
---------- -------- --------- ------ ------------ ---------------
mydata       9.0 MB sa             5 May 27, 2005  no options set

device_fragments  size    usage      created             free kbytes
----------------  ------  ---------- -----------         -----------
bigdevice         5.0 MB  data only  May 25 2005 3:42PM   3650
logdev          4.0 MB  log only   May 25 2005 3:42PM   not applicable
------------------------------------------------------------------------
-----
log only free kbytes = 4078
device                 segment
```

```
---------------------- ----------------------
bigdevice              default
bigdevice              system
logdev                 logsegment
(return status = 0)
```

Like all newly created databases, `mydata` has the segments named `default`, `system`, and `logsegment`. Because **create database** used **log on**, the `logsegment` is mapped to its own device, `logdev`, and the `default` and `system` segments are both mapped to `bigdevice`.

If you add space on the same database devices to `mydata`, and run **sp_helpdb** again, you see entries for the added fragments:

```
use master
alter database mydata on bigdevice = "2M"
    log on logdev = "1M"
use mydata
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created       status
---------- -------- --------- ------ ------------  ---------------
mydata     12.0 MB  sa             4 May 25, 2005  no options set

device_fragments  size     usage     created              free kbytes
----------------- -------- ---------- -----------------      -------
----
bigdevice        5.0 MB   data only  May 25 2005 3:42PM          2048
logdev            4.0 MB   data only  May 25 2005 3:42PM     not
applicable
data only        2.0 MB   log only   May 25 2005 3:55PM          2040
log only          1.0 MB   log only   May 25 2005 3:55PM     not
applicable
------------------------------------------------------------------
--------
log only free kybytes = 5098
device                 segment
---------------------- ----------------------
bigdevice              default
bigdevice              system
logdev                 logsegment
```

Always add log space to log space and data space to data space. SAP ASE instructs you to use **with override** if you try to allocate a segment that is already in use for data to the log, or vice versa. Remember that segments are mapped to entire devices, and not just to the space fragments. If you change any of the segment assignments on a device, you make the change for all of the fragments.

The following example allocates a new database device that has not been used by `mydata`:

```
use master
alter database mydata on newdevice = 3
use mydata
sp_helpdb mydata
```

```
name       db_size  owner     dbid   created      status
---------- -------- --------- ------ ------------ ---------------
mydata      15.0 MB sa             5 May 25, 2005  no options set

device_fragments   size     usage      created           free kbytes
------------------ --------- ---------- ------------------  -------
-------
bigdevice           5.0 MB    data only   May 25 2005
3:42PM             3650
logdev              4.0 MB    log only    May 25 2005 3:42PM  not
applicable
bigdevice           2.0 MB    data only   May 25 2005
3:55PM             2040
logdev              1.0 MB    log only    May 25 2005 3:55PM  not
applicable
newdevice           3.0 MB    data only   May 26 2005
11:59AM            3060
------------------------------------------------------------------
---------
log only free kbytes = 5098
device            segment
----------------- ----------
bigdevice         default
bigdevice         system
logdev            logsegment
newdevice         default
newdevice         system
```

The following example creates a segment called `new_space` on `newdevice`:

```
sp_addsegment new_space, mydata, newdevice
```

Here is the portion of the **sp_helpdb** report which lists the segment mapping:

```
device                          segment
------------------------------ ------------------
bigdevice                       default
bigdevice                       system
logdev                          logsegment
newdevice                       default
newdevice                       new_space
newdevice                       system
```

The `default` and `system` segments are still mapped to `newdevice`. If you are planning to use `new_space` to store a user table or index for improved performance, and you want to ensure that other user objects are not stored on the device by default, reduce the scope of `default` and `system` with **sp_dropsegment**:

```
sp_dropsegment system, mydata, newdevice
```

```
sp_dropsegment "default", mydata, newdevice
```

You must include the quotes around “default”; it is a Transact-SQL reserved word.

Here is the portion of the **sp_helpdb** report that shows the segment mapping:

```
device                          segment
------------------------------ --------------------
bigdevice                       default
bigdevice                       system
logdev                          logsegment
newdevice                       new_space
```

Only `new_space` is now mapped to `newdevice`. Users who create objects can use **on** `new_space` to place a table or index on the device that corresponds to that segment. Since the `default` segment is not pointing to that database device, users who create tables and indexes without using the **on** clause are not placing them on your specially prepared device.

If you use **alter database** on `newdevice` again, the new space fragment acquires the same segment mapping as the existing fragment of that device (that is, the `new_space` segment only).

At this point, if you use **create table** and name `new_space` as the segment, you get results like these from **sp_helpsegment**:

```
create table mytabl (c1 int, c2 datetime)
    on new_space
```

```
sp_helpsegment new_space
```

```
segment    name                            status
------- ------------------------------    ------
      3    new_space                            0

device           size        free_pages
---------------- ----------- -----------
newdevice        3.0MB              1523

Objects on segment 'new_space':

table_name        index_name         indid        partition_name
----------------- ------------------ --------     ----------------
mytabl            mytabl                    0     mytabl_400001425

Objects currently bound to segment 'new_space':

table_name      index_name     indid

total_size    total_pages    free_pages  used_pages   reserved_pages
------------ -----------    ----------- -----------   --------------
3.0MB                1536           1523          13                0
```

CHAPTER 11 Using the reorg Command

Update activity against a table can eventually lead to inefficient utilization of space and reduced performance; use the **reorg** to command reorganize the use of table space, and improve performance.

reorg is useful when:

- A large number of forwarded rows causes extra I/O during read operations.
- Inserts and serializable reads are slow because they encounter pages with noncontiguous free space that must be reclaimed.
- Large I/O operations are slow because of low cluster ratios for data and index pages.
- **sp_chgattribute** has been used to change a space management setting (**reservepagegap**, **fillfactor**, or **exp_row_size**) and the change is to be applied to all existing rows and pages in a table, not just to future updates.

The **reorg** command includes four parameters for carrying out different types and levels of reorganization:

- **reorg forwarded_rows** undoes row forwarding.
- **reorg reclaim_space** reclaims unused space left on a page as a result of deletions and row-shortening updates.
- **reorg compact** both reclaims space and undoes row forwarding.
- **reorg rebuild** undoes row forwarding, reclaims unused page space, and:
  - Rewrites all rows to accord with a table's clustered index, if it has one
  - Rewrites space for data and index partitions.
  - Works on individual partitions.
  - Writes rows to data pages to accord with any changes made in space management settings through **sp_chgattribute**
  - Drops and re-creates all indexes belonging to the table

Consider the following before running **reorg rebuild**:

- **reorg rebuild** holds an exclusive table lock for its entire duration. On a large table this may be a significant amount of time. However, **reorg rebuild** does everything that dropping and re-creating a clustered index does and takes less time. In addition, **reorg rebuild** rebuilds the table using all of the table's current space management settings. Dropping and re-creating an index does not use the space management setting for **reservepagegap**.
- In most cases, **reorg rebuild** requires additional disk space equal to the size of the table it is rebuilding and its indexes.

The following restrictions hold:

- The table specified in the command, if any, must use the datarows locking or datapages locking scheme.
- Versions of SAP ASE earlier than 15.0 restricted you from using **reorg rebuild** on allpages-locked tables. SAP ASE versions 15.0 and later allow you to run **reorg rebuild** on entire tables that uses allpages locking. **reorg rebuild** rebuilds the entire table, copying the data to new sets of pages, and rebuilding all indexes.
- You must be a system administrator or the object owner to issue **reorg**.
- You cannot issue **reorg** within a transaction.

reorg Command and Its Parameters

The reorg command includes a number of parameters.

reorg is useful when:

- A large number of forwarded rows causes extra I/O during read operations.
- Inserts and serializable reads are slow because they encounter pages with noncontiguous free space that must be reclaimed.
- Large I/O operations are slow because of low cluster ratios for data and index pages.
- **sp_chgattribute** has been used to change a space management setting (**reservepagegap**, **fillfactor**, or **exp_row_size**) and the change is to be applied to all existing rows and pages in a table, not just to future updates.

The **reorg** command includes five parameters for carrying out different types and levels of reorganization:

- **reorg forwarded_rows** undoes row forwarding.
- **reorg reclaim_space** reclaims unused space left on a page as a result of deletions and row-shortening updates.
- **reorg compact** both reclaims space and undoes row forwarding.
- **reorg defrag** allows you to schedule and resume reorganization, also allowing concurrent reads or writes on the data being reorganized.
- **reorg rebuild** undoes row forwarding, reclaims unused page space, and:
  - Rewrites all rows to accord with a table's clustered index, if it has one
  - Rewrites space for data and index partitions.
  - Works on individual partitions.
  - Writes rows to data pages to accord with any changes made in space management settings through **sp_chgattribute**
  - Drops and re-creates all indexes belonging to the table

Consider the following before running **reorg rebuild**:

- **reorg rebuild** holds an exclusive table lock for its entire duration. On a large table this may be a significant amount of time. However, **reorg rebuild** does everything that dropping and

re-creating a clustered index does and takes less time. In addition, **reorg rebuild** rebuilds the table using all of the table's current space management settings. Dropping and re-creating an index does not use the space management setting for **reservepagegap**.

- In most cases, **reorg rebuild** requires additional disk space equal to the size of the table it is rebuilding and its indexes.

The following restrictions apply:

- The table specified in **reorg** commands—excluding **reorg rebuild**—must have a datarows or datapages locking scheme.
- You must be a system administrator or the object owner to issue **reorg**.
- You cannot issue **reorg** within a transaction.

Running reorg rebuild Concurrently

The **reorg rebuild** command includes an **online** parameter that lets you reorganize data and perform maintenance without taking the data offline.

For example, to rebuild the indexes on the titles table and keep the data online, enter:

```
reorg rebuild titles with online
```

reorg rebuild ... online includes three phases:

- A blocking phase that takes exclusive table locks for a short duration to set up new metadata.
- A nonblocking phase that reorganizes the data and synchronizes the concurrent activity.
- A blocking phase that reacquires exclusive table locks to synchronize the remaining concurrent activity and install the new metadata.

SAP ASE recommends that you run **reorg rebuild ...online** when the table's transaction load is relatively low.

Running **reorg rebuild ... online** takes place in a single transaction. Recovering the work performed by the online parameter is similar to recovering the work performed without the online parameter. Consider these issues when rolling back work performed with the online parameter:

- Runtime rollback of the utility deallocates the pages allocated by the online parameter.
- Crash recovery clears the allocation status of the extents allocated by the online parameter, and makes them available for other tasks.
- In a high availability environment during node-failover recovery, if **reorg rebuild ... online** attempts to initiate a physical or logical lock, it waits for the recovery to complete before acquiring the lock.

reorg rebuild ... online includes these restrictions:

- Tables on which you run **reorg rebuild ... online** must have a unique index.
- All DMLs—that is, **select** (but not **select into**), **insert**, **update**, and **delete**—can operate on a table while **reorg rebuild ... online** is running. SAP ASE does not allow inserts that

lead to page splits on tables with all-pages locking scheme while **reorg rebuild ... online** is running.

- You cannot run more than one instance of **reorg rebuild ... online** simultaneously on a table.
- **reorg rebuild ... online** does not materialize non-nullable nonmaterialized default columns.

Using the optdiag Utility to Assess the Need for a reorg

To assess the need for running **reorg**, use statistics from the `systabstats` table and the **optdiag** utility.

`systabstats` contains statistics on the utilization of table space, while **optdiag** generates reports based on statistics in both `systabstats` and the `sysstatistics` table.

For information on the `systabstats` table, see *Performance and Tuning Series: Physical Database Tuning > Statistics Tables and Displaying Statistics with optdiag*. For information about **optdiag**, see the *Utility Guide*.

Moving Forwarded Rows to Home Pages

If an update makes a row too long to fit on its current page, the row is forwarded to another page.

A reference to the row is maintained on its original page, the row's home page, and all access to the forwarded row goes through this reference. Thus, it always takes two page accesses to get to a forwarded row. If a scan needs to read a large number of forwarded pages, the I/Os caused by extra page accesses slow performance.

reorg forwarded_rows undoes row forwarding by either moving a forwarded row back to its home page, if there is enough space, or by deleting the row and reinserting it in a new home page. If the table spans partitions, you can specify the partition with the *partition_name* parameter.

You can display statistics on the number of forwarded rows in a table by querying `systabstats` and using **optdiag**.

The syntax for **reorg forwarded_rows** is:

```
reorg forwarded_rows table_name partition partition_name
    [with {resume, time = no_of_minutes}]
```

reorg forwarded_rows does not apply to indexes, because indexes do not have forwarded rows.

Use reorg compact to Remove Row Forwarding

reorg forwarded_rows uses allocation page hints to find forwarded rows.

Because it does not have to search an entire table, this command executes quickly, but it may miss some forwarded rows. After running **reorg forwarded_rows**, you can evaluate its effectiveness by using **optdiag** and checking "Forwarded row count." If "Forwarded row count" is high, you can then run **reorg compact**, which goes through a table page by page and undoes all row forwarding.

See also

- *Reclaiming Unused Space and Undoing Row Forwarding* on page 214

Reclaiming Unused Space from Deletions and Updates

When a task performs a **delete** operation, or an update that shortens row length, the empty space is preserved in case the transaction is rolled back; unreclaimed space may accumulate to the point that it impairs performance.

reorg reclaim_space reclaims unused space left by deletions and updates. On each page that has space resulting from committed deletion or row-shortening updates, **reorg reclaim_space** rewrites the remaining rows contiguously, leaving all the unused space at the end of the page. If there are no remaining rows, **reorg reclaim_space** deallocates the page.

If the table extends over a partition, or several partitions, reclaim any available space on the partition by specifying *partition_name*.

You can display statistics on the number of unreclaimed row deletions in a table from the `systabstats` table and by using the **optdiag** utility. There is no direct measure of how much unused space there is as a result of row-shortening updates.

If you specify only a table name, only the table's data pages are reorganized to reclaim unused space; in other words, indexes are not affected. If you specify an index name, only the pages of the index are reorganized. If you specify a partition, only the part of the table that resides on that partition is affected.

Reclaming Space Without the reorg Command

SAP ASE includes a number of activities that reclaim or reorganize space in a table on a page-by-page basis.

- Inserts, when encountering a page that would have enough room if it reclaimed unused space
- The **update statistics** command (for index pages only)
- Re-creating clustered indexes

- The housekeeper garbage collection task, if **enable housekeeper GC** is set to **1** or more

Each of these has limitations and may be insufficient for use on a large number of pages. For example, inserts may execute more slowly when they need to reclaim space, and may not affect many pages with space that can be reorganized. Space reclamation under the housekeeper garbage collection task compacts unused space, but a single housekeeper garbage collection task that runs at user priority may not reach every page that needs it.

Reclaiming Unused Space and Undoing Row Forwarding

reorg compact combines the functions of **reorg reclaim_space** and **reorg forwarded_rows**.

Use **reorg compact** when:

- You do not need to rebuild an entire table (**reorg rebuild**); however, both row forwarding and unused space from deletions and updates may be affecting performance.
- There are a large number of forwarded rows.

If you specify a partition, only the part of the table that resides on that partition is affected.

See also

- *Use reorg compact to Remove Row Forwarding* on page 213

Rebuilding a Table

Issue **reorg rebuild** to rewrite all rows in a table to new pages, so the table is arranged according to its clustered index (if one exists).

Use **reorg rebuild** when:

- Large I/O is not being selected for queries where it is usually used, and **optdiag** shows a low cluster ratio for data pages, data rows, or index pages.
- You used **sp_chgattribute** to change one or more of the **exp_row_size**, **reservepagegap**, or **fillfactor** space management settings and you want the changes to apply not only to future data, but also to existing rows and pages. For information about **sp_chgattribute**, see the *Reference Manual: Procedures*.

If a table needs to be rebuilt because of a low cluster ratio, it may also need to have its space management settings changed.

If **reorg rebuild** finds that the current table is used by another session, it aborts the entire transaction.

reorg rebuild uses a table's current space management settings to rewrite the rows in the table according to the table's clustered index, if it has one. All indexes on the table are dropped and re-created using the current space management values for **reservepagegap** and **fillfactor**. After a **rebuild**, a table has no forwarded rows and no unused space from deletions or updates.

When you run it against a table and a partition, **reorg rebuild** performs the following:

1. Takes an exclusive table lock
2. Copies data from old to new pages
3. Deallocates old data pages
4. Locks system tables for updates (including `sysindexes`, `sysobjects`, `syspartitions`, and `systabstats`)
5. Rebuilds clustered and nonclustered indexes against new data pages
6. Commits all open transactions
7. Releases locks on system tables

If the table is large and has several indexes, the locks for updating system tables can be held for a long time and may block processes from accessing information in the system tables for the user tables on which you are running **reorg**. However, `systabstats` does not impact this blocking because it is already datarow-locked.

reorg rebuild builds the clustered index using the **with sorted data** option, so the data does not have to be re-sorted during this index build.

Prerequisites for Running reorg rebuild

You must perform some preliminary steps before running reorg rebuild.

- Set the database option **select into/bulkcopy/pllsort** to **true**.
- Determine which locking scheme your table uses. You can issue **reorg** against an entire allpages-locked tables. However, you cannot issue fine grained **reorg index** or **reorg partition level** on allpages-locked tables.
- Make sure that additional disk space, equal to the size of the table and its indexes, is available.

To set **select into/bulkcopy/pllsort** to **true**, enter:

```
1>use master
2> go
1> sp_dboption pubs2, "select into/bulkcopy/pllsort", true
2> go
```

Following a **rebuild** on a table:

- You must dump the database containing the table before you can dump the transaction log.
- Distribution statistics for the table are updated.
- All stored procedures that reference the table are recompiled the next time they are run.

Changing Space Management Settings Before Using reorg rebuild

If it appears that a table quickly becomes fragmented and must be rebuilt too frequently, you may need to change the table's space management settings before you run **reorg rebuild**. When **reorg rebuild** rebuilds a table, it rewrites all table and index rows according to the table's current settings for **reservepagegap**, **fillfactor**, and **exp_row_size**.

If it appears that a table quickly becomes fragmented and must be rebuilt too frequently, you may need to change the table's space management settings before you run **reorg rebuild**.

These properties all affect how quickly inserts cause a table to become fragmented, as measured by a low cluster ratio.

Use **sp_chgattribute** to change the space management settings (see the *Reference Manual: Procedures*). For reference information about **sp_chgattribute**, see the *Reference Manual: Commands*, and for additional detailed information about space management, see *Performance and Tuning Series: Physical Database Tuning > Setting Space Management Properties*.

Using the reorg rebuild Command on Indexes

The **reorg rebuild** command allows you to rebuild indexes, while the table itself remains accessible for read and update activities.

Rebuilding a single table or partition index rewrites all index rows to new pages, which improves performance by:

- Improving clustering of the leaf level of the index
- Applying stored values for the fillfactor on the index, which can reduce page splits
- Applying any stored value for **reservepagegap**, which can help reserve pages for future splits

To reduce contention with users whose queries must use the index, **reorg rebuild** locks a small number of pages at a time. Rebuilding an index is a series of independent transactions, with some independent, nested transactions. Approximately 32 pages are rebuilt in each nested transaction, and approximately 256 pages are rebuilt in each outer transaction. Address locks are acquired on the pages being modified and are released at the end of the top action. The pages deallocated in a transaction are not available for reuse until the next transaction begins.

If the **reorg rebuild** command stops running, the transactions that are already committed are not rolled back. Therefore, the part that has been reorganized is well-clustered with desired space utilization, and the part that has not been reorganized is the same as it was before you ran the command. The index remains logically consistent.

Note: Rebuilding the clustered index does not affect the data pages of the table. It only affects the leaf pages and higher index levels. Non-leaf pages above level 1 are not rebuilt.

Rebuilding Indexes with reorg rebuild index_name partition_name

Rebuilding a single table or partition index rewrites all index rows to new pages.

This improves performance by:

- Improving clustering of the leaf level of the index

- Applying stored values for the fillfactor on the index, which can reduce page splits
- Applying any stored value for **reservepagegap**, which can help reserve pages for future splits

To reduce contention with users whose queries must use the index, **reorg rebuild** locks a small number of pages at a time. Rebuilding an index is a series of independent transactions, with some independent, nested transactions. Approximately 32 pages are rebuilt in each nested transaction, and approximately 256 pages are rebuilt in each outer transaction. Address locks are acquired on the pages being modified and are released at the end of the top action. The pages deallocated in a transaction are not available for reuse until the next transaction begins.

If the **reorg rebuild** command stops running, the transactions that are already committed are not rolled back. Therefore, the part that has been reorganized is well-clustered with desired space utilization, and the part that has not been reorganized is the same as it was before you ran the command. The index remains logically consistent.

Note: Rebuilding the clustered index does not affect the data pages of the table. It only affects the leaf pages and higher index levels. Non-leaf pages above level 1 are not rebuilt.

Space Requirements for Rebuilding an Index

If you do not specify **fill_factor** or **reservepagegap**, rebuilding an index requires additional space of approximately 256 pages, or less in the data segment.

The amount of log space required is larger than that required to drop the index and re-create it using **create index**, but it should be only a small fraction of the actual index size. The more additional free space is available, the better the index clustering will be.

Running **reorg rebuild *table_name*** updates the statistics for all leading index columns. However, running **reorg rebuild *table_name index_name*** does not automatically update the statistics. Instead, SAP ASE automatically updates index statistics when you run **reorg rebuild *index_name*** if the update includes a suffiecient change in data to affect its plan choice and performance.

Note: reorg rebuild may not rebuild those parts of the index that are already well clustered and use an acceptable space utilization.

Status Messages

Running **reorg rebuild** *indexname* on a large table may take a long time. Periodic status messages appear; starting and ending messages are written to the error log and to the client process that is executing **reorg**. In-progress messages appear only on the client.

A status reporting interval is calculated as either 10 percent of the pages to be processed or 10,000 pages, whichever is larger. When this number of pages is processed, a status message prints. Therefore, no more than 10 messages are printed, regardless of the size of the index. Status messages for existing **reorg** commands print more frequently.

resume and time Options for Reorganizing Large Tables

Use the **resume** and **time** options of the **reorg** command when reorganizing an entire table would take too long and interfere with other database activities.

time allows you to run a **reorg** for a specified length of time. **resume** allows you to start a **reorg** at the point in a table where the previous **reorg** finished. In combination, the two options allow you to reorganize a large table by running a series of partial reorganizations (for example, during offhours).

resume and **time are** not available with **reorg rebuild**.

Specifying no_of_minutes in the time Option

The *no_of_minutes* argument in the **time** option refers to elapsed time, not CPU time.

For example, to run **reorg compact** for 30 minutes, beginning where a previous **reorg compact** finished, enter:For example, to run **reorg compact** for 30 minutes, beginning where a previous **reorg compact** finished, enter:

```
reorg compact tablename with resume, time=30
```

If the **reorg** process goes to sleep during any part of the 30 minutes, it still counts as part of the elapsed time and does not add to the duration of the **reorg**.

When the amount of time specified has passed, **reorg** saves statistics about the portion of the table or index that was processed in the `systabstats` table. This information is used as the restart point for a **reorg** with the **resume** option. The restart points for each of the three parameters that take **resume** and **time** options are maintained separately. You cannot, for example, start with **reorg reclaim_space** and then resume the process using **reorg compact**.

If you specify *no_of_minutes*, and **reorg** arrives at the end of a table or an index before the time is up, it returns to the beginning of the object and continues until it reaches its time limit.

Note: resume and **time** allow you to reorganize an entire table or index over multiple runs. However, if there are updates between **reorg** runs, some pages may be processed twice and some pages may not be processed at all.

Incremental Reorganization

The **defrag** parameter for the **reorg** command allows you schedule and resume reorganization, while also allowing concurrent reads or writes on the data being reorganized.

reorg defrag executes multiple reorganization transactions one right after the other, compared with traditional **reorg**, which reorganizes the entire table data in a single transaction. You can specify a time interval during which it reorganizes the data. This allows other processes to run concurrently without affecting reorganization.**reorg defrag** locks each row or page, as per the

locking scheme of the object, and begins and commits a transaction for every data chunk processed. The data reorganization space requirement does not exceed the size of an allocation unit (256 data pages). Index updates for the reorganized data do not consume extra space. For every data partition undergoing incremental reorganization, a row is stored in `sysattributes`.

Checking the Reorganization Status

Use information from **sp_helpdefrag** or the **defrag_status** built-in function to decide if reorganization must be resumed on a table or if it should start from the beginning.

Additionally, **sp_helpdefrag** and **defrag_status()** can also be used to track the status of reorganization of the table when **reorg defrag** is in progress.

sp_helpdefrag and **defrag_status** give the percentage of the table or partition reorganized and whether the reorganization is in progress on the table or in an intermittent state to resume.

You must execute **sp_helpdefrag** in the context of the database where the table resides. If you do not specify a table name, output includes all of the tables in the database which have valid information about defragmentation status.

The syntax for **sp_helpdefrag** is:

```
sp_helpdefrag [table_name][, partition_name]
```

The syntax for **defrag_status** is:

```
defrag_status( dbid, objid [ , ptnid | -1 [, "tag" ] ]
```

Clearing reorg defrag Information from sysattributes

For every data partition undergoing incremental reorganization, a row is stored in `sysattributes`. Use the **dbcc** command **zapdefraginfo** to delete this information from `sysattributes` before performing a downgrade.

Also in a running server, if the rows with defragmentation information for a specific object are accidentally lost from `sysattributes` due to unknown reasons, use **dbcc zapdefraginfo** to reset the extent version information for the specific object so that a later **reorg defrag** will not fail to consider all the extents of the object.

Logging Behavior

Default incremental reorganization logging includes the extent allocation, extent version update, index row deletion, index row insertion, extent deallocation, and scan position update, in that order.

This logging sequence ensures complete recoverability of the reorganization. However, this sequence may not enforce the **dump tran sequence**.

Turn on either **full logging for reorg rebuild** or the general **full logging for all** options in the existing database to ensure that the incremental reorganization enforces the **dump tran**

sequence, an extra log of the page image is required for every destination data page created by the reorganization.

The default logging behavior consumes the following log space by **reorg defrag**:

- Number of extent allocation log records –
  - Max value: (32 / 'number of preallocated extents').
  - Min value: 1.
- Number of extent version update log records – Number of the destination extents allocated in this transaction (Maximum 32).
- Total number of index delete/insert log records – 2 x (number of indexes on the table) x (number of data rows de-fragmented in this transaction).
- Number of extent deallocation log records –1 (for all the extents deallocated in this transaction).
- Number of page deallocation log records – Number of unused destination pages in the transaction.
- Number of scan position update log records – 1 (for the last scan position of this transaction).
- Extra logging to achieve full logging behavior:
  - Number of page image log records – Number of used destination pages in the transaction (Maximum 255).

CHAPTER 12 Checking Database Consistency

The database consistency checker (**dbcc**) provides commands for checking the logical and physical consistency of a database.

dbcc checks:

- Page linkage and data pointers at both the page level and the row level using **checkstorage** or **checktable** and **checkdb**
- Page allocation using **checkstorage**, **checkalloc**, **checkverify**, **tablealloc**, **textalloc**, and **indexalloc**
- For consistency within and between the system tables in a database with **checkcatalog**

dbcc checkstorage stores the results of checks in the `dbccdb` database. You can print reports from `dbccdb` using the **dbcc** stored procedures.

Use the **dbcc** commands:

- As part of regular database maintenance—the integrity of the internal structures of a database depends upon the system administrator or database owner running database consistency checks on a regular basis.
- To determine the extent of possible damage after a system error has occurred.
- Before backing up a database for additional confidence in the integrity of the backup.
- If you suspect that a database is damaged. For example, if using a particular table generates the message "Table corrupt," you can use **dbcc** to determine if other tables in the database are also damaged.

If you are using Component Integration Services, there are additional **dbcc** commands you can use for remote databases. See the *Component Integration Services Users Guide*.

See also

- *Conditions That Do Not Disable Mirroring* on page 21

Page and Object Allocation

When you initialize a database device, the **disk init** command divides the new space into *allocation units*.

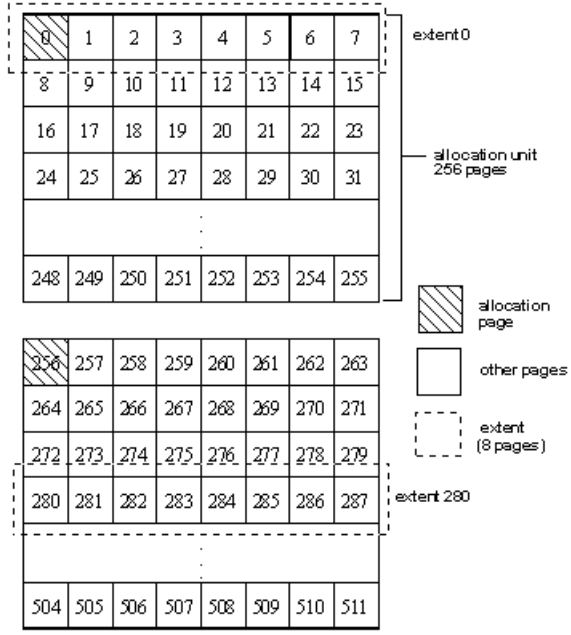
The size of the allocation unit depends on the size of the logical pages your server uses (2, 4, 8, or 16K). The first page of each allocation unit is an *allocation page*, which tracks the use of all pages in the allocation unit. Allocation pages have an object ID of 99; they are not real database objects and do not appear in system tables, but **dbcc** errors on allocation pages report this value.

When a table of an indexed partition requires space, SAP ASE allocates a block of 8 pages to the object. This 8-page block is called an extent. Each allocation unit contains 32 extents. The size of the extent also depends on the size of the server logical pages. SAP ASE uses extents as a unit of space management to allocate and deallocate space as follows:

- When you create a table of an index partition, SAP ASE allocates an extent for the object.
- When you add rows to an existing table, and the existing pages are full, SAP ASE allocates another page. If all pages in an extent are full, SAP ASE allocates an additional extent.
- When you drop a table of an indexed partition, SAP ASE deallocates the extents it occupied.
- When you delete rows from a table so that it shrinks by a page, SAP ASE deallocates the page. If the table shrinks off the extent, SAP ASE deallocates the extent.

Every time space is allocated or deallocated on an extent, SAP ASE records the event on the allocation page that tracks the extents for that object. This provides a fast method for tracking space allocations in the database, since objects can shrink or grow without excess overhead.

This figure shows how data pages are set up within extents and allocation units in SAP ASE databases.



dbcc checkalloc checks all allocation pages (page 0 and all pages divisible by 256) in a database and reports on the information it finds. **dbcc indexalloc** and **dbcc tablealloc** check the allocation of specific database objects.

Understanding the Object Allocation Map (OAM)

Each table and index on a table has an *object allocation map* (*OAM*), which holds information on pages allocated to the table or index and is checked when a new page is needed for the index or table.

The OAM is stored on pages allocated to the table or index and is checked when a new page is needed for the index or table. A single OAM page can hold allocation mapping for between 2,000 and 63,750 data or index pages. Each OAM page is the size of one logical page size. For example, on a server using a logical page size of 4K, each OAM page is 4K.

The number of entries per OAM page also depends on the logical page size the server is using. The following table describes the number of OAM entries for each logical page size:

2K Logical Page Size	4K Logical Page Size	8K Logical Page Size	16K Logical Page Size
250	506	1018	2042

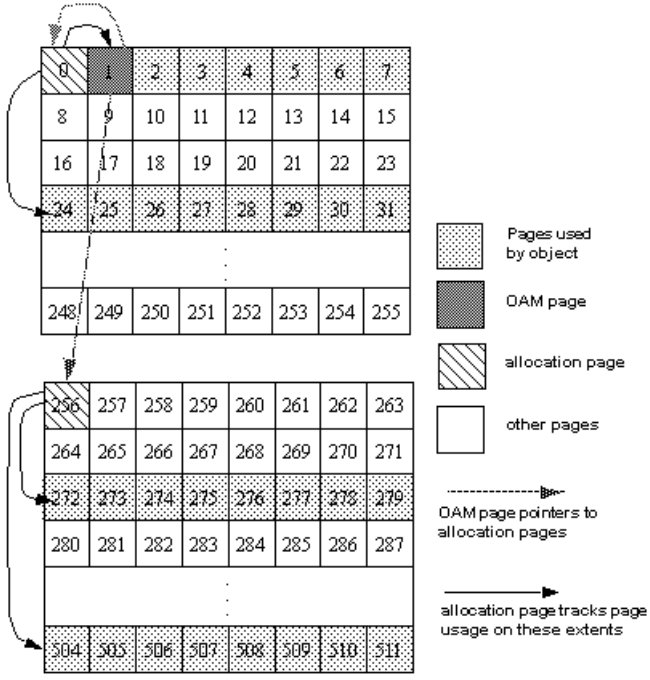
The OAM pages point to the allocation page for each allocation unit where the object uses space. The allocation pages, in turn, track the information about extent and page usage within the allocation unit. In other words, if the `titles` table is stored on extents 24 and 272, the OAM page for the `titles` table points to pages 0 and 256.

The figure below shows an object stored on 4 extents, numbered 0, 24, 272 and 504 for a server that uses 2K logical pages. The OAM is stored on the first page of the first segment. In this case, since the allocation page occupies page 0, the OAM is located on page 1.

This OAM points to two allocation pages: page 0 and page 256.

These allocation pages track the pages used in each extent used by all objects with storage space in the allocation unit. For the object in this example, it tracks the allocation and deallocation of pages on extents 0, 24, 272, and 504.

Figure 17: OAM Page and Allocation Page Pointers



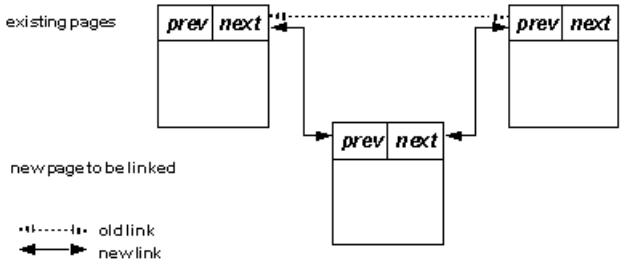
dbcc checkalloc and **dbcc tablealloc** examine this OAM page information, in addition to checking page linkage.

Understanding Page Linkage

After a page has been allocated to a table of an indexed partition, it is linked with other pages that are used for the same object.

Each page contains a header that includes the number of the page that precedes it ("prev") and of the page that follows it ("next"). When a new page is allocated, the header information on the surrounding pages changes to point to that page. **dbcc checktable** and **dbcc checkdb** check page linkage. **dbcc checkalloc**, **tablealloc**, and **indexalloc** compare page linkage to information on the allocation page.

Figure 18: How a newly allocated page is linked with other pages



dbcc Checks

The **dbcc** commands hat you can run depend on your role.

For example:

- Only the table owner can execute **dbcc** with the **checktable**, **fix_text**, or **reindex** keywords.
- Only the database owner can use the **checkstorage**, **checkdb**, **checkcatalog**, **checkalloc**, **indexalloc**, **textalloc**, and **tablealloc** keywords.
- Only a system administrator can use the **dbrepair** keyword.

Table 6. Comparison of checks performed by dbcc commands

Checks Performed	checkstorage	checktable	checkdb	checkalloc	indexalloc	tablealloc	checkcatalog	textalloc
Allocation of text valued columns	X							X[1]
Index consistency		X	X					
Index sort order		X	X					
OAM page entries	X	X	X	X	X	X		X
Page allocation	X			X	X	X		X

Checks Performed	checkstorage	checktable	checkdb	checkalloc	indexalloc	tablealloc	checkcatalog	textalloc
Page consistency	X	X	X					
Pointer consistency	X	X	X					
System tables							X	X[2]
Text column chains	X	X	X					X
Text valued columns	X	X	X					X[3]

[1] **textalloc** does not check the allocation status for data pages holding columns, but does check the allocation status for text pages.

[2] **textalloc** checks the `syspartition` entry corresponding to the text or image column.

[3] **textalloc** checks:

- The data page that holds the text valued columns
- The text page in which it saves the text values.

Note: You can run all **dbcc** commands except **dbrepair** and **checkdb** with the **fix** option while the database is active.

dbcc Command Output

dbcc checkstorage stores its results in the `dbccdb` database. You can print a variety of reports from this database.

The output of most other **dbcc** commands includes information that identifies the objects being checked, and error messages that indicate any problems the command finds in the object. When you run **dbcc tablealloc** and **dbcc indexalloc** with **fix**, the output also indicates the repairs that the command makes.

The following example shows **dbcc tablealloc** output for a table with an allocation error:

```
dbcc tablealloc(rrtab)
```

```
The default report option of OPTIMIZED is used for this run.
The default fix option of FIX   is used for this run.
```

```
*****************************************************************
TABLE: rrtab            OBJID = 416001482
PARTITION ID=432001539 FIRST=2032 ROOT=2040 SORT=1
Data level: indid 1, partition 432001539. 2 Data pages allocated and
2 Extents
allocated.
Indid : 1, partition : 432001539. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=448001596 FIRST=2064 ROOT=2072 SORT=1
Data level: indid 1, partition 448001596. 2 Data pages allocated and
2 Extents
allocated.Indid : 1, partition : 448001596. 1 Index pages allocated
and 2 Extents
allocated.
PARTITION ID=480001710 FIRST=2080 ROOT=2080 SORT=0
Indid : 2, partition : 480001710. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=496001767 FIRST=2096 ROOT=2096 SORT=0
Indid : 2, partition : 496001767. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=512001824 FIRST=2112 ROOT=2112 SORT=0
Indid : 3, partition : 512001824. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=528001881 FIRST=2128 ROOT=2128 SORT=0
Indid : 3, partition : 528001881. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=544001938 FIRST=680 ROOT=680 SORT=0
Indid : 4, partition : 544001938. 1 Index pages allocated and 2
Extents
allocated.
TOTAL # of extents = 18
Alloc page 1792 (# of extent=2 used pages=2 ref pages=2)
Alloc page 1792 (# of extent=2 used pages=3 ref pages=3)
Alloc page 1792 (# of extent=1 used pages=1 ref pages=1)
Alloc page 2048 (# of extent=1 used pages=1 ref pages=1)
Alloc page 1792 (# of extent=1 used pages=1 ref pages=1)
Alloc page 2048 (# of extent=1 used pages=2 ref pages=2)
Alloc page 2048 (# of extent=2 used pages=3 ref pages=3)
Alloc page 2048 (# of extent=2 used pages=2 ref pages=2)
Alloc page 2048 (# of extent=2 used pages=2 ref pages=2)
Alloc page 2048 (# of extent=2 used pages=2 ref pages=2)
Alloc page 256 (# of extent=1 used pages=1 ref pages=1)
Alloc page 512 (# of extent=1 used pages=1 ref pages=1)
Total (# of extent=18 used pages=21 ref pages=21) in this database
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role..
```

Checking Database and Table Consistency

The **dbcc** commands for checking the consistency of databases and tables are **dbcc checkstorage**, **dbcc checktable**, **dbcc checkindex**, and **dbcc checkdb**.

dbcc checkstorage

dbcc checkstorage runs checks against the database on disk.

Use **dbcc checkstorage** to check:

- Allocation of text valued columns
- Page allocation and consistency
- OAM page entries
- An OAM page for each partition exists
- Pointer consistency
- Text-valued columns and text-column chains

If a corruption is only in memory, **dbcc checkstorage** may not detect the corruption. To ensure consistency between two **dbcc checkstorage** runs, execute **checkpoint** before running **dbcc checkstorage**. However, doing this may turn a transient memory corruption into a disk corruption.

The advantages of using **dbcc checkstorage** are:

- Combines many of the checks provided by the other **dbcc** commands
- Does not lock tables or pages for extended periods, which allows **dbcc** to locate errors accurately while allowing concurrent update activity
- Scales linearly with the aggregate I/O throughput
- Separates the functions of checking and reporting, which allows custom evaluation and report generation
- Provides a detailed description of space usage in the target database
- Records **dbcc checkstorage** activity and results in the `dbccdb` database, which allows trend analysis and provides a source of accurate diagnostic information

dbcc checkstorage is different from the other **dbcc** commands in that it requires:

- The `dbccdb` database to store configuration information and the results of checks made on the target database. However, you can run **dbcc checkstorage** from any database.
- At least two workspaces to use during the check operation. See “dbccdb Tables” in the *Reference Manual: Tables*.
- System and stored procedures to help you prepare your system to use **dbcc checkstorage** and to generate reports on the data stored in `dbccdb`.

dbcc checkstorage does not repair any faults. After you run **dbcc checkstorage** and generate a report to see the faults, you can run the appropriate **dbcc** command to repair the faults.

Understanding the dbcc checkstorage Operation

The **dbcc checkstorage** operation consists of a series of steps.

1. Inspection – **dbcc checkstorage** uses the device allocation and the segment definition of the database being checked, and the **max worker processing** and **named cache** configuration parameters to determine the level of parallel processing that can be used. **dbcc checkstorage** also uses the configuration parameters **max worker processes** and **dbcc named cache** to limit the level of parallel processing that can be used.
2. Planning – **dbcc checkstorage** generates a plan for executing the operation that takes advantage of the parallelism discovered in step 1.
3. Execution and optimization – **dbcc checkstorage** uses SAP ASE worker processes to perform parallel checking and storage analysis of the target database. It attempts to equalize the work performed by each worker process and consolidates the work of under-utilized worker processes. As the check operation proceeds, **dbcc checkstorage** extends and adjusts the plan generated in step 2 to take advantage of the additional information gathered during the check operation.
4. Reporting and control – during the check operation, **dbcc checkstorage** records in the `dbccdb` database all the faults it finds in the target database for later reporting and evaluation. It also records the results of its storage analysis in `dbccdb`. When **dbcc checkstorage** encounters a fault, it attempts to recover and continue the operation, but ends operations that cannot succeed after the fault. For example, a defective disk does not cause **dbcc checkstorage** to fail; however, check operations performed on the defective disk cannot succeed, so they are not performed.

   If another session performs **drop table** concurrently, **dbcc checkstorage** might fail in the initialization phase. If this happens, run **dbcc checkstorage** again when the drop table process is finished.

Note: See the *Troubleshooting and Error Messages Guide* for information about **dbcc checkstorage** error messages.

Performance and Scalability

dbcc checkstorage scales linearly with aggregate I/O throughput for a substantial performance improvement over **dbcc checkalloc**.

The scaling property of **dbcc checkstorage** means that if the database doubles in size and the hardware doubles in capacity (realizable I/O throughput), the time required for a **dbcc** check remains unchanged. Doubling the capacity would typically mean doubling the number of disk spindles and providing sufficient additional I/O channel capacity, system bus capacity, and CPU capacity to realize the additional aggregate disk throughput.

Most of the checks performed by using **dbcc checkalloc** and **dbcc checkdb**, including text column chain verification, are achieved with a single check when you use **dbcc checkstorage**, thereby eliminating redundant check operations.

dbcc checkstorage checks the entire database, including unused pages, so execution time is relative to database size. Therefore, when you use **dbcc checkstorage**, there is not a large difference between checking a database that is nearly empty and checking one that is nearly full, as there is with the other **dbcc** commands.

Unlike the other **dbcc** commands, the performance of **dbcc checkstorage** does not depend heavily on data placement. Therefore, performance is consistent for each session, even if the data placement changes between sessions.

Because **dbcc checkstorage** does extra work to set up the parallel operation and records large amounts of data in `dbccdb`, the other **dbcc** commands are faster when the target database is small.

The location and allocation of the workspaces used by **dbcc checkstorage** can affect performance and scalability. See *Reference Manual: Tables > dbccdb Tables*.

To run **dbcc checkstorage** and one of the system procedures for generating reports with a single command, use **sp_dbcc_runcheck**.

dbcc checktable

dbcc checktable performs a series of checks on the specified table.

dbcc checktable verifies that:

- Index and data pages are linked correctly
- Indexes are sorted properly
- Pointers are consistent
- All indexes and data partitions are correctly linked
- Data rows on each page have entries in the row-offset table; these entries match the locations for the data rows on the page
- Partition statistics for partitioned tables are correct

The **skip_ncindex** option allows you to skip checking the page linkage, pointers, and sort order on nonclustered indexes. The linkage and pointers of clustered indexes and data pages are essential to the integrity of your tables. You can drop and re-create nonclustered indexes if SAP ASE reports problems with page linkage or pointers.

partition_name is the name of the partition you are checking (this may or may not contain the entire table because tables can span multiple partitions), and *partition_id* is the ID of the partition you are checking.

If you specify *partition_name* or *partition_id*, **dbcc checktable** checks only the table, or parts of the table, residing on this partition; it does not expand its check to other partitions, and has the following restrictions:

- If the table consists of more than one partition, index processing is limited to local indexes.
- If you specify the *partition_name* or *partition_id* parameter, you must also specify either the second parameter (**skip_ncindex** or **fix_spacebits**) or null. This example specifies null:

```
dbcc checkalloc(titles, null, 560001995)
```

- If the sort order or character set for a table with columns defined with `char` or `varchar` datatypes is incorrect, **dbcc checktable** does not corrct these values. You must run **dbcc checktable** on the entire table to correct these errors.
- If an index is marked "read-only" due to a change in the sort order, **dbcc checktable** does not clear the O_READONLY bit in the `sysstat` field for the table's `sysobjects` entry. To clear this status bit, run **dbcc checktable** on the entire table.
- If you run **dbcc checktable** on `syslogs`, **dbcc checktable** does not report space usage (free space versus used space). However, if you do not specify *partition_name* or *partition_id* parameters, **dbcc checktable** reports the space usage.

When **checkstorage** returns a fault code of 100035, and **checkverify** confirms that the spacebit fault is a hard fault, you can use **dbcc checktable** to fix the reported fault.

The following command checks part of the `titles` table that resides on the `smallsales` partition (which contains all book sales less than 5000):

```
dbcc checktable(titles, NULL, "smallsales")
```

```
Checking partition 'smallsales' (partition ID 1120003990) of table
'titles'.
The logical page size of this table is 8192 bytes. The total number
of data
pages in partition 'smallsales' (partition ID 1120003990) is 1.
Partition 'smallsales' (partition ID 1120003990) has 14 data rows.
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

You can use **dbcc checktable** with the table name or the table's object ID. The `sysobjects` table stores this information in the `name` and `id` columns.

The following example shows a report on an undamaged table:

```
dbcc checktable(titles)
```

```
Checking table 'titles' (object ID 576002052):Logical page size is
8192 bytes.
The total number of data pages in partition 'titleidind_576002052'
(partition ID
576002052) is 1.
The total number of data pages in this table is 1.
Table has 18 data rows.
DBCC execution completed. If DBCC printed error messages, contact a
user with
System Administrator (SA) role.
```

To check a table that is not in the current database, supply the database name. To check a table owned by another object, supply the owner's name. You must enclose any qualified table name in quotes. For example:

```
dbcc checktable("pubs2.newuser.testtable")
```

dbcc checktable addresses the following problems:

- If the page linkage is incorrect, **dbcc checktable** displays an error message.
- If the sort order (`sysindexes.soid`) or character set (`sysindexes.csid`) for a table with columns with `char` or `varchar` datatypes is incorrect, and the table's sort order is compatible with SAP ASE default sort order, **dbcc checktable** corrects the values for the table. Only the binary sort order is compatible across character sets.

  Note: If you change sort orders, character-based user indexes are marked read-only and must be checked and rebuilt, if necessary.
- If data rows are not accounted for in the first OAM page for the object, **dbcc checktable** updates the number of rows on that page. This is not a serious problem. The built-in function **row_count** uses this value to provide fast row estimates in procedures like **sp_spaceused**.

You can improve **dbcc checktable** performance by using enhanced page fetching.

dbcc checkindex

dbcc checkindex runs the same checks as **dbcc checktable**, but only on the specified index instead of the entire table.

partition_name is the name of the partition you are checking and *partition_id* is the ID of the partition you are checking. **bottom_up** specifies that **checkindex** checks from the bottom up, starting at the leaf of the index. **bottom_up** is only applicable for dataonly locked tables. If you specify this option with **checkindex** or **checktable**, the index checking is done in a bottom-up fashion

dbcc checkdb

dbcc checkdb runs the same checks as **dbcc checktable** on each table in the specified database.

If you do not specify a database name, **dbcc checkdb** checks the current database. **dbcc checkdb** produces messages similar to those returned by **dbcc checktable**, and makes the same types of corrections.

If you specify the optional **skip_ncindex**, **dbcc checkdb** does not check any of the nonclustered indexes on user tables in the database.

If the database extends over a series of partitions, **dbcc checkdb** performs its checks on each partition.

Checking Page Allocation

The **dbcc** commands that you use to check page allocation are **dbcc checkalloc**, **dbcc indexalloc**, **dbcc tablealloc**, and **dbcc textalloc**.

dbcc checkalloc

If you do not provide a database name, **dbcc checkalloc** checks the current database.

dbcc checkalloc ensures that:

- All pages are correctly allocated
- Partition statistics on the allocation pages are correct
- Every page that is allocated is used
- All pages are correctly allocated to individual partitions
- Only allocated pages are used

With the **fix** option, **dbcc checkalloc** fixes all allocation errors that would otherwise be fixed by **dbcc tablealloc**, and also fixes pages that remain allocated to objects that have been dropped from the database. Before you can use **dbcc checkalloc** with the **fix** option, you must put the database into single-user mode.

dbcc checkalloc output consists of a block of data for each table, including the system tables, and the indexes on each table. For each table or index, it reports the number of pages and extents used. The INDID values are:

- Tables without clustered indexes have an INDID=0.
- For allpages locking tables with clustered indexes, the table data partitions and clustered index partitions are consolidated, with an INDID=1 for the data partitions (or the clustered index partitions).
- For dataonly locked tables with clustered index, the table data partitions have an INDID=0. The clustered index and nonclustered indexes are numbered consecutively, starting with INDID=2.

Partition and page information is listed after PARTITION ID=*partition_number*.

The following report on `pubs2` shows output for the `titleauthor`, `titles`, and `stores` tables:

```
************************************************************
TABLE: titleauthor OBJID = 544001938
PARTITION ID=544001938 FIRST=904 ROOT=920 SORT=1
Data level: indid 1, partition 544001938. 1 Data pages allocated and
2 Extents
allocated.
Indid : 1, partition : 544001938. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=544001938 FIRST=928 ROOT=928 SORT=0
Indid : 2, partition : 544001938. 1 Index pages allocated and 2
Extents
allocated.
PARTITION ID=544001938 FIRST=944 ROOT=944 SORT=0
Indid : 3, partition : 544001938. 1 Index pages allocated and 2
Extents
allocated.
```

```
TOTAL # of extents = 8
************************************************************
TABLE: titles OBJID = 576002052
PARTITION ID=1120003990 FIRST=1282 ROOT=1282 SORT=1
Data level: indid 0, partition 1120003990. 1 Data pages allocated and
1 Extents
allocated.
PARTITION ID=1136004047 FIRST=1289 ROOT=1289 SORT=1
Data level: indid 0, partition 1136004047. 1 Data pages allocated and
1 Extents
allocated.
TOTAL # of extents = 2
************************************************************
TABLE: stores OBJID = 608002166
PARTITION ID=608002166 FIRST=745 ROOT=745 SORT=0
Data level: indid 0, partition 608002166. 1 Data pages allocated and
1 Extents
allocated.
TOTAL # of extents = 1
```

dbcc indexalloc

dbcc indexalloc is an index-level version of **dbcc checkalloc**, providing the same integrity checks on an individual index, including checking for index partitions.

dbcc indexalloc checks the specified index to see that:

- All pages are correctly allocated.
- Every page that is allocated is used
- Only allocated pages are used

You must specify an object ID, an object name, or a partition ID, and you must also specify an index ID. **dbcc checkalloc** and **dbcc indexalloc** output includes index IDs.

To use the **fix** or **nofix** option for **dbcc indexalloc**, you must specify one of the report options (**full**, **optimized**, **fast**, or **null**). If you specify a partition ID, only that partition is checked.

dbcc indexalloc treats unpartitioned indexes as indexes with a single partition.

You can run **sp_indsuspect** to check the consistency of sort order in indexes, and **dbcc reindex** to repair inconsistencies.

See also

- *Correcting Allocation Errors Using the fix | nofix Options* on page 235
- *Generate Reports with dbcc tablealloc and dbcc indexalloc* on page 236

dbcc tablealloc

dbcc tablealloc requires that you specify the table name, the data partition ID, or the table's object ID from the ID column in `sysobjects`.

dbcc tablealloc checks the specified user table to ensure that:

- All pages are correctly allocated.
- Partition statistics on the allocation pages are correct.
- Every page that is allocated is used
- Only allocated pages are used
- All pages are correctly allocated to the partitions in the specified table.

If you specify a data partition ID, **dbcc tablealloc** performs its checks on this partition and all local index partitions. If you specify an object name or an object ID, **dbcc tablealloc** performs its checks on the entire table. If you specify an index partition ID, **dbcc tablealloc** returns error 15046.

To use the **fix** or **nofix** options for **dbcc tablealloc**, you must specify one of the report options (**full**, **optimized**, **fast**, or **null**).

dbcc textalloc

dbcc textalloc checks the allocation integrity of an object's text and image data, and reports and fixes any problems it finds.

dbcc textalloc checks the specified object—or all objects that have text or image data in the current database—to ensure that:

- Text and image entries in the system tables are correct.
- OAM page chain and allocation of OAM pages for text and image data are correct.
- All text and image pages are correctly allocated.
- No text page is allocated but not used.
- No text page is used but not allocated.

You can issue **dbcc textalloc** without any parameters. By default, **dbcc textalloc** runs in default mode. If you do not specify a parameter, **dbcc textalloc** checks each table in the current database that has a text or image column.

You can run **dbcc textalloc** concurrently on different objects without interfering with each other.

Correcting Allocation Errors Using the fix | nofix Options

The **fix** | **nofix** option with **dbcc checkalloc**, **dbcc tablealloc**, **textalloc**, and **dbcc indexalloc** specifies whether or not the command fixes the allocation errors in tables. The default for user tables is **fix**. The default for system tables is **nofix**.
Before you can use the **fix** option on system tables, you must put the database into single-user mode:

```
sp_dboption dbname, "single user", true
```

You can issue this command only when no one is using the database.

Output from **dbcc tablealloc** with **fix** displays allocation errors and any corrections that are made. This is an example of an error message that appears whether or not the **fix** option is used:

```
Msg 7939, Level 22, State 1:
Line 2:
Table Corrupt: The entry is missing from the OAM for object id
144003544 indid 0 for allocation page 2560.
```

When you use **fix**, this message indicates that the missing entry has been restored:

```
The missing OAM entry has been inserted.
```

The **fix | nofix** option works the same in **dbcc indexalloc** as it does in **dbcc tablealloc**.

See also

- *dbcc indexalloc* on page 234

Generate Reports with dbcc tablealloc and dbcc indexalloc

Generate **full**, **optimized**, or **fast** reports with **dbcc tablealloc** or **dbcc indexalloc**.

- **full** – produces a report containing all types of allocation errors. Using the **full** option with **dbcc tablealloc** gives the same results as using **dbcc checkalloc** at a table level.
- **optimized** – produces a report based on the allocation pages listed in the OAM pages for the table. When you use the **optimized** option, **dbcc tablealloc** does not report and cannot fix unreferenced extents on allocation pages that are not listed in the OAM pages. If you do not specify a report type, or if you specify **null**, **optimized** is the default.
- **fast** – produces an exception report of pages that are referenced but not allocated in the extent (2521-level errors); does not produce an allocation report.

See also

- *dbcc indexalloc* on page 234

Checking Consistency of System Tables

dbcc checkcatalog checks for consistency within and between the system tables in a database.

For example, it verifies that:

- Every type in `syscolumns` has a matching entry in `systypes`.
- Every table and view in `sysobjects` has at least one column in `syscolumns`.
- `sysindexes` is consistent, and fixes any errors
- For each row in `syspartitions`, there is a matching row in `sysegments`.

- The last checkpoint in `syslogs` is valid.

It also lists the segments defined for use by the database and fixes any errors it finds.

If you do not specify a database name, **dbcc checkcatalog** checks the current database:

```
dbcc checkcatalog (testdb)
```

```
Checking testdb
The following segments have been defined for database 5
(database name testdb).
virtual start addr      size      segments
--------------------    ------    --------------------------
33554432                4096       0
                                   1
16777216                102        2
DBCC execution completed. If DBCC printed error messages, see your
System
Administrator.
```

Strategies for Using Consistency Checking Commands

dbcc checkdb, **dbcc checktable**, and **dbcc checkcatalog** perform different types of integrity checks than **dbcc checkalloc**, **dbcc tablealloc**, and **dbcc indexalloc**.

dbcc checkstorage performs a combination of the some of the checks performed by the other commands. This table shows which checks are performed by the commands.

Command and Option	Level	Locking and Performance	Speed	Thoroughness
checkstorage	Page chains and data rows for allocation pages, page linkages for indexes, OAM pages, device and partition statistics	No locking; performs extensive I/O and may saturate the system's I/O; can use dedicated cache with minimal impact on other caches	Fast	High
checktable checkdb	Page chains, sort order, data rows, and partition statistics for all indexes	Shared table lock; **dbcc checkdb** locks one table at a time and releases the lock after it finishes checking that table	Slow	High
checktable checkdb with **skip_ncindex**	Page chains, sort order, and data rows for tables and clustered indexes	Shared table lock; **dbcc checkdb** locks one table at a time and releases the lock after it finishes checking that table	Up to 40 percent faster than without **skip_ncindex**	Medium

Command and Option	Level	Locking and Performance	Speed	Thoroughness
checkalloc	Page chains and partition statistics	No locking; performs extensive I/O and may saturate the I/O calls; only allocation pages are cached	Slow	High
tablealloc , **indexalloc** , and **textalloc** with **full**	Page chains	Shared table lock; performs extensive I/O; only allocation pages are cached	Slow	High
tablealloc , **indexalloc**, and **textalloc** with **optimized**	Allocation pages	Shared table lock; performs extensive I/O; only allocation pages are cached	Moderate	Medium
tablealloc , **indexalloc**, and **textalloc** with **fast**	OAM pages	Shared table lock	Fast	Low
checkcatalog	Rows in system tables	Shared page locks on system catalogs; releases lock after each page is checked; very few pages cached	Moderate	Medium

Using Large I/O and Asynchronous Prefetch

Some **dbcc** commands can use large I/O and asynchronous prefetch if you configured these options for the caches that are used by the databases or objects to be checked.

dbcc checkdb and **dbcc checktable** use large I/O pools for the page chain checks on tables when the tables use a cache with large I/O configured. The largest available I/O size is used. When checking indexes, **dbcc** uses only 2K buffers.

The **dbcc checkdb**, **dbcc checktable**, and the **dbcc** allocation checking commands, **checkalloc**, **tablealloc** and **indexalloc**, use asynchronous prefetch when it is available for the pool in use. See *Performance and Tuning Series: Query Processing and Abstract Plans > Setting Limits for dbcc*.

Cache-binding commands and the commands to change the size and asynchronous prefetch percentages for pools are dynamic commands. If you use these **dbcc** commands during off-peak periods, when user applications experience little impact, you can change these settings to speed **dbcc** performance and restore the normal settings when **dbcc** checks are finished.

Scheduling Database Maintenance at Your Site

Several factors, including database use, backup schedule, table size, and importance of data, determine how often you should run **dbcc** commands, and which ones to run.

Database Use

Consider your overall database when determining which **dbcc** commands to run, and when to run them.

If your SAP ASE is used primarily between the hours of 8:00 a.m. and 5:00 p.m., Monday through Friday, you can run **dbcc** checks at night and on weekends so that the checks do not have a significant impact on your users.

If your tables are not extremely large, you can run a complete set of **dbcc** commands fairly frequently.

dbcc checkstorage and **dbcc checkcatalog** provide the best coverage at the lowest cost, and assure recovery from backups. You can run **dbcc checkdb** or **dbcc checktable** less frequently to check index sort order and consistency. You need not coordinate this check with any other database maintenance activity. Reserve object-level **dbcc** checks and those checks that use the **fix** option for further diagnosis and correction of faults found by **dbcc checkstorage**.

If your SAP ASE is used 24 hours a day, 7 days a week, you may want to limit the resource usage of **dbcc checkstorage** by limiting the number of worker processes, or by using application queues. If you decide not to use **dbcc checkstorage**, you may want to schedule a cycle of checks on individual tables and indexes using **dbcc checktable**, **dbcc tablealloc**, and **dbcc indexalloc**. At the end of the cycle, when all tables have been checked, you can run **dbcc checkcatalog** and back up the database. See “Distributing Engine Resources” in the *Performance and Tuning Series: Basics*.

Some sites with 24-hour, high-performance demands run **dbcc** checks by:

- Dumping the database to tape
- Loading the database dump into a separate SAP ASE to create a duplicate database
- Running **dbcc** commands on the duplicate database
- Running **dbcc** commands with the **fix** options on appropriate objects in the original database, if errors are detected that can be repaired with the **fix** options

The dump is a logical copy of the database pages; therefore, problems found in the original database are present in the duplicate database. This strategy is useful if you are using dumps to provide a duplicate database for reporting or some other purpose.

Schedule **dbcc** commands that lock objects to run when they avoid interference with business activities. For example, **dbcc checkdb** acquires locks for each table on which it performs the database check and then releases the lock once it finishes and procedes to the next table. These tables are not accessible while **dbcc checkdb** holds the lock. Do not schedule **dbcc checkdb** (or other **dbcc** commands with similar side effects) to run while other business activities require the tables that are locked.

Backup Schedule

The more often you back up your databases and dump your transaction logs, the more data you can recover after a failure. You must decide how much data you are willing to lose in the event of a disaster, and develop a dump schedule to support that decision.

After you schedule your dumps, decide how to incorporate the **dbcc** commands into that schedule. You are not required to perform **dbcc** checks before each dump; however, you may lose additional data if a corruption occurs while the dump is taking place.

An ideal time to dump a database is after you run a complete check of that database using **dbcc checkstorage** and **dbcc checkcatalog**. If these commands find no errors in the database, you know that your backup contains a clean database. You can reindex to correct problems that occur after loading the dump. Use **dbcc tablealloc** or **indexalloc** on individual tables and indexes to correct allocation errors reported by **dbcc checkalloc**.

Size of Tables and Importance of Data

dbcc checkstorage is a database-level operation. If only a few tables contain critical data or data that changes often, you may want to run the table- and index-level **dbcc** commands more frequently on those tables than you run **dbcc checkstorage** on the entire database.

Before you run a **dbcc** command, answer the following questions about your data:

- How many tables contain highly critical data?
- How often does that data change?
- How large are those tables?

Errors Generated by Database Consistency Problems

Errors generated by database consistency problems encountered by **dbcc checkstorage** are documented in the `dbcc_types` table.

Most of the error numbers are in the ranges 5010 – 5024 and 100,000 – 100,038. For information on specific errors, see *Reference Manual: Tables > dbccdb Tables*.

Errors generated by database consistency problems encountered by **dbcc** commands other than **dbcc checkstorage** usually have error numbers from 2500 – 2599 or from 7900 – 7999. These messages, and others that can result from database consistency problems (such as Error 605), may include phrases like “Table Corrupt” or “Extent not within segment.”

Some messages indicate severe database consistency problems; others are not so urgent. A few may require help from SAP Technical Support, but most can be solved by:

- Running **dbcc** commands that use the **fix** option
- Following the instructions in the *Error Messages and Troubleshooting Guide*, which contains step-by-step instructions for resolving many **dbcc** database errors.

Whatever techniques are required to solve the problems, the solutions are much easier when you find the problem soon after the occurrence of the corruption or inconsistency. Consistency

problems can exist on data pages that used infrequently, such as a table that is updated only monthly. **dbcc** can find, and often fix, these problems for you.

Reporting on Aborted checkstorage and checkverify Operations

When a **checkstorage** or **checkverify** operation aborts, it prints a message that contains the operation ID (`opid`) and the name of the database that was being examined when the operation aborted.

An aborted **checkverify** operation also provides a sequence number in the message, which instructs the user to run **sp_dbcc_patch_finishtime**, with the provided *dbname*, `opid`, and (if it was a **checkverify** operation), the sequence number, *seq*. After executing **sp_dbcc_patch_finishtime**, you can create fault reports on the aborted operation.

Aborting with Error 100032

checkstorage may abort an object check when it encounters a page linkage error (100032).

checkstorage continues to verify the object if a more recent version of the page eliminates the page linkage error, or if the number of page linkage errors is fewer than the configured maximum linkage error value.

Use **sp_dbcc_updateconfig** to configure the maximum linkage error value. This example configures the `great_big_db` with a value of 8:

```
sp_dbcc_updateconfig great_big_db, "linkage error abort", "8"
```

See *Reference Manual: Building Blocks > dbcc Stored Procedures*.

checkstorage may abort its check before reaching the page linkage error when:

- Concurrent updates on objects that use APL indexes disrupt the page linkage because **checkstorage** may not be able to access the rest of the page chain
- An index is dropped during the page linkage check

Running **checkstorage** in a quiet state should reduce (or eliminate) the transient errors that lead to an aborted index check. To eliminate transient faults, run **checkverify** immediately after running **dbcc checkstorage**.

Comparison of Soft and Hard Faults

When **dbcc checkstorage** finds a fault in the target database, it is recorded in the `dbcc_faults` table as either a *soft fault* or a *hard fault*.

Soft Faults

A *soft fault* is an inconsistency in SAP ASE that is usually not persistent.

Most soft faults result from temporary inconsistencies in the target database caused by user updates to the database during **dbcc checkstorage** or when **dbcc checkstorage** encounters data definition language (DDL) commands. These faults are not repeated when you run the command a second time. You can reclassify soft faults by comparing the results of the two

executions of **dbcc checkstorage** or by running **dbcc tablealloc** and **dbcc checktable** after **dbcc checkstorage** finds soft faults.

If the same soft faults occur in successive executions of **dbcc checkstorage**, they are "persistent" soft faults, and may indicate a corruption. If you execute **dbcc checkstorage** in single-user mode, the soft faults reported are persistent. You can resolve these faults by using **sp_dbcc_differentialreport**, or by running **dbcc tablealloc** and **dbcc checktable**. If you use the latter two commands, you need to check only the tables or indexes that exhibited the soft faults.

Hard Faults

A *hard fault* is a persistent corruption of SAP ASE that cannot be corrected by restarting SAP ASE.

Not all hard faults are equally severe. For example, each of the following situations cause a hard fault, but the results are different:

- A page that is allocated to a nonexistent table minimally reduces the available disk storage.
- A table with some rows that are unreachable by a scan might return the wrong results.
- A table that is linked to another table causes the query to stop.

Some hard faults can be corrected by simple actions such as truncating the affected table. Others can be corrected only by restoring the database from a backup.

Verifying Faults with dbcc checkverify

dbcc checkverify examines the results of the most recent **checkstorage** operation (by reading **dbcc_faults**) and reclassifies each soft fault as either a hard fault or an insignificant fault. **checkverify** acts as a second filter to remove spurious faults from the **checkstorage** results.

checkverify reports only the suspect faults, and resolves their issues or reports them as soft faults or as harmless using a procedure similar to the procedure used by the **checkstorage** operation. **checkverify** does not read hard faults because they already classified as hard by **checkstorage**.

Note: checkverify locks the table against concurrent updates, which ensures that the soft faults are reclassified correctly. **checkverify** does not find errors that have occurred since the last run of **checkstorage**.

checkverify records information in the `dbcc_operation_log` and `dbcc_operation_results` tables the same way that **checkstorage** does. The recorded value of `opid` is the same as the `opid` of the last **checkstorage** operation. **checkverify** updates the `status` column in the `dbcc_faults` table and inserts a row in the `dbcc_fault_params` table for the faults it processes.

checkverify does not use the `scan` or `text` workspaces.

checkverify verifies each suspect fault **checkstorage** finds as hard or soft. In this example, **checkstorage** finds two hard faults, and three suspect (soft) faults (`faults` means hard faults, `suspect condtions` means soft faults):

```
dbcc checkstorage(victimdb)
Checking victimdb: Logical pagesize is 2048 bytes
DBCC CHECKSTORAGE for database 'victimdb' sequence 1 completed at Jun
20 2013
1:32PM. 2 faults and 3 suspect conditions were located. 0 checks were
aborted.
You should investigate the recorded faults, and plan a course of
action that
will correct them.
Suspect conditions are to be treated as faults if the same suspect
condition
persists in subsequent CHECKSTORAGE operations, or if they are also
detected
by other DBCC functions.
```

Run dbcc checkverify to determine if the suspect conditions are hard or soft faults:

```
dbcc checkstorage(victimdb)
Verifying faults for 'victimdb'.
DBCC CHECKVERIFY for database 'victimdb' sequence 1 completed at Jun
20 2013
1:33PM. 3 suspect conditions considered, 3 identified as faults, 0
identified
as harmless, and 0 could not be checked. 0 objects could not be
checked.
```

The `3 suspect conditions considered` listed in the output are the same 3 reported by **checkstorage** above. **checkverify** only verifies them, it does not fix them. If the faults were soft faults, they would not need fixing. However, all 3 suspect faults are identified as hard. The total number of hard faults is 5: 2 found in **checkstorage** + 3 that were upgraded from soft faults by **checkverify**

Run **sp_dbcc_faultreport** to verify the 5 faults:

```
sp_dbcc_faultreport "short", "victimdb"
Database Name : victimdb
Table Name   Index   PartitionId Fault Type                     Page
Number
------------ ------- ----------------- -------------------------
-----
ALLOCATION   0       99              100030 (page format error)  768
foo          0       624002223        100017 (OAM ring error)     864
foo          0       624002223        100022 (chain start error)  865
foo          0       624002223        100031 (page not allocated) 864
foo           0        624002223          100031 (page not allocated)
865
```

Each fault found by **checkstorage** is verified by **checkverify** as one of the following:

- A hard fault classified as such by **checkstorage**.

- A soft fault reclassified as hard by **checkverify** because concurrent activity was ruled out as the cause.
- A soft fault confirmed to be soft by **checkverify**. Some soft faults that appear when there is no concurrent activity in the database do not represent a significant hazard and are not reclassified as hard. A soft fault is not reclassified if it is informational only and not a corruption.
- A soft fault reclassified as insignificant because it can be attributed to concurrent activity or because subsequent activity masked the original inconsistency.

A fault that is assigned code 100011 (text pointer fault) by **checkstorage** is verified as hard if the text column has a hard fault. If it does not, it is reclassified as soft.

A fault that is assigned code 100016 (page allocated but not linked) by **checkstorage** is verified as hard if the same fault appears in two successive **checkstorage** operations. Otherwise, it is reclassified as soft.

When a fault that is assigned code 100035 (spacebits mismatch) by **checkstorage** is verified as hard, you can repair it by using **dbcc checktable**.

When **checkverify** confirms hard faults in your database, follow the appropriate procedures to correct the faults.

checkverify classifies the following fault codes as soft faults:

- 100020 – check aborted.
- 100025 – row count fault.
- 100028 – page allocation off current segment.

Scheduling dbcc checkverify

Verify persistent faults by running **checkverify** anytime after running **checkstorage**, even after an extended period of hours or days. However, when deciding your schedule, keep in mind that the database state changes over time, and the changes can mask both soft faults and hard faults.

For example, a page that is linked to a table but not allocated is a hard fault. If the table is dropped, the fault is resolved and masked. If the page is allocated to another table, the fault persists but its signature changes. The page now appears to be linked to two different tables. If the page is reallocated to the same table, the fault appears as a corrupt page chain.

Persistent faults that are corrected by a subsequent database change do not usually pose operational problems. However, detecting and quickly verifying these faults may locate a source of corruption before more serious problems are encountered, or before the signature of the original fault changes. For this reason, SAP recommends that you run **checkverify** as soon as possible after running **dbcc checkstorage**.

Note: When you execute **checkstorage** with the target database in single-user mode, there are no transient soft faults and therefore, no need to execute **checkverify**.

checkverify runs only one time for each execution of **checkstorage**. However, if **checkverify** is interrupted and does not complete, you can run it again. The operation resumes from where it was interrupted.

checkverify may take a long time to complete when processing very large databases. During this time, **checkverify** does not provide you with any indication of when it will finish.

To see progress status reports during **checkverify**, use the **command_status_reporting** command:

```
set command_status_reporting on
```

Now, when you run **checkverify**, you see results similar to:

```
dbcc checkverify (pubs2)
```

```
Verifying faults for 'pubs2'.
Verifying faults for table 't1'. The total number of tables to verify
is 5. This
is table number 1.
Verifying faults for table 't2'. The total number of tables to verify
is 5. This
is table number 2.
Verifying faults for table 't3'. The total number of tables to verify
is 5. This
is table number 3.
Verifying faults for table 't4'. The total number of tables to verify
is 5. This
is table number 4.
Verifying faults for table 't5'. The total number of tables to verify
is 5. This
is table number 5.
DBCC CHECKVERIFY for database 'pubs2' sequence 4 completed at Apr  9
2003
2:40PM. 72 suspect conditions were resolved as faults, and 11 suspect
conditions
were resolved as harmless. 0 objects could not be checked.
```

Executing dbcc checkverify

checkverify operates on the results of the last completed **checkstorage** operation for the specified database only.

When the **checkverify** operation is complete, SAP ASE returns:

```
DBCC checkverify for database name, sequence
n completed at date time. n suspect conditions resolved as faults
and n resolved as innocuous.
n checks were aborted.
```

You can run **checkverify** automatically after running **checkstorage** by using **sp_dbcc_runcheck**.

You can exclude tables, faults, and table or fault combinations from **checkverify**. Use **sp_dbcc_exclusions** to indicate which items you want excluded from **checkverify**. **sp_dbcc_exclusions** is dynamic; that is, **checkverify** immediately excludes the items you

specify in **sp_dbcc_exclusions**. Once you exclude these objects from **checkverify**, it does not report on them the next time you run the command.

Preparing to Use dbcc checkstorage

Before you can use **dbcc checkstorage**, configure SAP ASE and set up the `dbccdb` database.

Each action is described in detail in the following sections. The examples in this section assume a server that uses 2K logical pages.

This table summarizes the steps and commands in the order you should use them.

Table 7. Preparing to use dbcc checkstorage

Action	Use
1. Obtain recommendations for database size, devices (if `dbccdb` does not exist), workspace sizes, cache size, and the number of worker processes for the target database.	**use master** **sp_plan_dbccdb**
2. If necessary, adjust the number of worker processes that SAP ASE uses.	**sp_configure** **number of worker processes** **memory per worker processes**
3. (Optional) Create a named cache for **dbcc**.	**sp_cacheconfig**
4. Configure your buffer pool.	**sp_poolconfig**
5. If `dbccdb` already exists, drop it and all associated devices before creating a new `dbccdb` database.	**drop database**
6. Initialize disk devices for the `dbccdb` data and the log.	**disk init**
7. (Optional) Create `dbccdb` on the data disk device.	**create database**
8. (Optional) Add disk segments.	**use dbccdb**
9. Populate the `dbccdb` database and install **dbcc** stored procedures.	`isql -Usa -P -i $SYBASE/$SYB-ASE_ASE/scripts /installdbccdb`

Note: dbcc checkstorage runs its checks against the database on disk. If a corruption exists only in memory, **dbcc** may not detect it. To ensure consistency between two sequential **dbcc checkstorage** commands, first run a **checkpoint**. Be aware that running **checkpoint** may turn a transient memory corruption into a disk corruption.

Planning Resources

Selecting the appropriate device and size for dbccdb is critical to the performance of **dbcc checkstorage** operations.

sp_plan_dbccdb provides configuration recommendations for the specified target database depending on whether dbccdb exists or not. Use this information to configure SAP ASE and set up the dbccdb database.

If dbccdb does not exist, **sp_plan_dbccdb** returns:

- Minimum size for dbccdb
- Devices that are suitable for dbccdb
- Minimum sizes for the scan and text workspaces
- Minimum cache size
- Number of worker processes

The values recommended for the cache size are approximate, because the optimum cache size for dbccdb depends on the pattern of the page allocation in the target database. The following example from a server that uses 2K logical pages shows the output of **sp_plan_dbccdb** for the pubs2 database when dbccdb does not exist:

```
sp_plan_dbccdb pubs2
```

```
Recommended size for dbccdb is 4MB.

Recommended devices for dbccdb are:

Logical Device Name   Device Size   Physical Device Name

sprocdev              28672         /remote/SERV/sprocs_dat
tun_dat               8192          /remote/SERV/tun_dat
tun_log               4096          /remote/SERV/tun_log

Recommended values for workspace size, cache size and process count
are:

dbname         scan ws    text ws    cache    process count

pubs2          64K          64K        640K     1

(return status = 0)
```

If dbccdb already exists, **sp_plan_dbccdb** returns:

- Minimum size for dbccdb
- Size of existing dbccdb database
- Minimum sizes for the scan and text workspaces
- Minimum cache size
- Number of worker processes

This example shows the output of **sp_plan_dbccdb** for the `pubs2` database when `dbccdb` already exists:

```
sp_plan_dbccdb pubs2
```

```
Recommended size for dbccdb database is 23MB (data = 21MB, log =
2MB).
dbccdb database already exists with size 8MB.
Recommended values for workspace size, cache size and process count
are:
dbname                          scan ws    text ws    cache    process
count
pubs2                              64K        48K        640K     1
(return status = 0)
```

If you plan to check more than one database, use the name of the largest one for the target database. If you do not provide a target database name, **sp_plan_dbccdb** returns configuration values for all databases listed in `master..sysdatabases`:

```
sp_plan_dbccdb
```

```
Recommended size for dbccdb is 4MB.

dbccdb database already exists with size 8MB.

Recommended values for workspace size, cache size and process count
are:

dbname           scan ws    text ws    cache    process count

master            64K        64K        640K     1
tempdb            64K        64K        640K     1
model             64K        64K        640K     1
sybsystemprocs   384K       112K       1280K     2
pubs2             64K        64K        640K     1
pubs3             64K        64K        640K     1
pubtune          160K        96K       1280K     2
sybsecurity       96K        96K       1280K     2
dbccdb           112K        96K       1280K     2
```

See the *Reference Manual: Procedures*.

Planning Workspace Size

dbcc requires a scan workspace and a text workspace.

Space requirements for these workspaces depend on the size of the largest database you are checking. To run concurrent **dbcc checkstorage** operations, set up additional workspaces.

Different databases can use the same workspaces. Therefore, the workspaces must be large enough to accommodate the largest database with which they will be used.

Note: sp_plan_dbccdb suggests workspace sizes—the following details for determining the workspace size are provided for background information only.

Each page in the target database is represented by one 18-byte row in the `scan` workspace. This workspace should be approximately 1.1 percent of the target database size.

Every non-null `text` column in the target database inserts a 22-byte row in the `text` workspace. If there are *n* non-null `text` columns in the target database, the size of the `text` workspace must be at least (22 * *n*) bytes. The number of non-null `text` columns is dynamic, so allocate enough space for the `text` workspace to accommodate future demands. The minimum space required for the `text` workspace is 24 pages.

Default Scan and Text Workspaces

Default scan and text workspaces are created when you run the **installdbccdb** script.

The name of the default scan workspace is `def$scan$ws` and its size is 256K. The default text workspace is named `def$text$ws` and its size is 128K. These default workspaces are used if you have not configured default workspaces and the target database does not have configured workspace values.

Number of Concurrent Workspaces

You can configure `dbccdb` to run **dbcc checkstorage** concurrently on multiple databases.

However, this is possible only when the second and subsequent **dbcc checkstorage** operations have their own dedicated resources. To perform concurrent **dbcc checkstorage** operations, each operation must have its own `scan` and `text` workspaces, worker processes, and reserved cache.

The total space requirement for workspaces depends on whether user databases are checked serially or concurrently. If you run **dbcc checkstorage** operations serially, the largest `scan` and `text` workspaces can be used for all user databases. If you run **dbcc checkstorage** operations concurrently, set `dbccdb` to accommodate the largest workspaces that will be used concurrently. Determine the workspace sizes by adding the sizes of the largest databases that will be checked concurrently.

See the *Reference Manual: Tables*.

Automatic Workspace Expansion

The **sp_dbcc_updateconfig. . automatic workspace expansion** option indicates whether **checkstorage** automatically resizes the workspace, if necessary.

At the beginning of a **checkstorage** run, the size of the workspace is validated. If more space is needed, and **automatic workspace expansion** is enabled, **checkstorage** automatically expands the workspace if adequate space is available on the respective segments. If more space is needed, and **automatic workspace expansion** is not enabled, **checkstorage** exits and prints an informative message.

A value of 1 (the default value) enables **automatic workspace expansion**. A value of 0 disables **automatic workspace expansion**. See the *Reference Manual: Procedures*.

Configuring Worker Processes

The **max worker processes**, **number of worker processes**, and **memory per worker process** configuration parameters affect **dbcc checkstorage**.

- **max worker processes** – set this parameter with **sp_dbcc_updateconfig**. It updates the value of `max worker processes` in the `dbcc_config` table for each target database.
- **number of worker processes** – set this configuration parameter with **sp_configure**. It updates the `server_name.cfg` file.
- **memory per worker process** – set this configuration parameter with **sp_configure**. It updates the `server_name.cfg` file.

max worker processes specifies the maximum number of worker processes used by **dbcc checkstorage** for each target database, while **number of worker processes** specifies the total number of worker processes supported by SAP ASE. Worker processes are not dedicated to running **dbcc checkstorage** operations.

Set the value for **number of worker processes** high enough to allow for the number of processes specified by **max worker processes**. A low number of worker processes reduces the performance and resource consumption of **dbcc checkstorage**. **dbcc checkstorage** cannot use more processes than the number of database devices used by the database. Cache size, CPU performance, and device sizes might suggest a lower worker processes count. If there are not enough worker processes configured for SAP ASE, **dbcc checkstorage** does not run.

maximum parallel degree and **maximum scan parallel degree** have no effect on the parallel functions of **dbcc checkstorage**. When **maximum parallel degree** is set to 1, parallelism in **dbcc checkstorage** is not disabled.

dbcc checkstorage requires multiple processes, so **number of worker processes** must be set to at least 1 to allow for a parent process and a worker process.

sp_plan_dbccdb recommends values for the number of worker processes, depending on database size, number of devices, and other factors. You can use smaller values to limit the load on your system. **dbcc checkstorage** may use fewer worker processes than **sp_plan_dbccdb** recommends, or fewer than you configure.

Using more worker processes does not guarantee faster performance. The following scenario describes the effects of two different configurations:

- An 8GB database has 4GB of data on disk A and 0.5GB of data on each of the disks B, C, D, E, F, G, H, and I.
- With 9 worker processes active, the time it takes to run **dbcc checkstorage** is 2 hours, which is the time it takes to check disk A. Each of the other 8 worker processes finishes in 15 minutes and waits for the disk A worker process to finish.
- With 2 worker processes active, the time it takes to run **dbcc checkstorage** is still 2 hours. The first worker process processes disk A and the other worker process processes disks B,

C, D, E, F, G, H, and I. In this case, there is no waiting, and resources are used more efficiently.

memory per worker process specifies the total memory allocation for worker processes support in SAP ASE. The default value is adequate for **dbcc checkstorage**.

Setting a Named Cache for dbcc

If you use a named cache for **dbcc checkstorage**, you might need to adjust the SAP ASE configuration parameters.

During a **dbcc checkstorage** operation, the workspaces are temporarily bound to a cache which also reads the target database. Using a named cache that is dedicated to **dbcc** minimizes the impact of the database check on other users and improves performance. You can create a separate cache for each **dbcc checkstorage** operation that runs concurrently, or you can create one cache that is large enough to fit the total requirements of the concurrent operations. The size required for optimum performance depends on the size of the target database and distributions of data in that database. **dbcc checkstorage** requires a minimum of 640K of buffers (each buffer is one extent) per worker process in the named cache.

For best performance, assign most of the dedicated cache to the buffer pool and do not partition the cache. The recommended cache size is the minimum size for the buffer pool. Add the size of the one-page pool to this value.

If you dedicate a cache for **dbcc checkstorage**, the command does not require more than the minimum one-page buffer pool. If the cache is shared, you can improve the performance of **dbcc checkstorage** by increasing the buffer pool size before running the operation, and reducing the size after the operation is complete. The buffer pool requirements are the same for a shared cache. However, while a shared cache may meet the size requirement, other demands on the cache might limit the buffer availability to **dbcc checkstorage** and greatly impact the performance of both **checkstorage** and SAP ASE as a whole.

Note: Because SAP ASE may issue error messages 9946 or 9947 if you dedicate a cache partition for **dbcc checkstorage**, SAP recommends that you dedicate a named cache for **dbcc checkstorage**.

To configure SAP ASE with a named cache for **dbcc checkstorage** operations, use **sp_cacheconfig** and **sp_poolconfig**.

Configuring an 8-page I/O Buffer Pool

dbcc checkstorage requires a I/O buffer pool of one extent (8 data pages).

Use **sp_poolconfig** to configure the pool size and verify that the pool has been configured properly. The pool size is stored in the `dbcc_config` table.

The size of the **dbcc checkstorage** buffer pool is the page size multiplied by 16. The **dbcc checkstorage** buffer pool requirements for the different page sizes are:

- (2KB page server) * (8 extents) = 16K buffer pool
- (4KB page server) * (8 extents) = 32K buffer pool
- (8KB page server) * (8 extents) = 64K buffer pool
- (16KB page server) * (8 extents) = 128K buffer pool

The following example shows how to use **sp_poolconfig** to set a 16K buffer pool for "master_cache" on a server configured for 2K logical pages (the named cache is created for the `master` database):

```
sp_poolconfig "master_cache", "1024K", "16K"
```

The following example shows that the buffer pool for the private cache `master_cache` is set:

```
1> sp_poolconfig "master_cache"
2> go
```

```
Cache Name     Status    Type     Config Value Run Value
-------------  --------- -------- ------------ ------------
master_cache   Active    Mixed         2.00 Mb      2.00 Mb
                                    ------------ ------------
                          Total        2.00 Mb      2.00 Mb
================================================================
Cache: master_cache,   Status: Active,   Type: Mixed
   Config Size: 2.00 Mb,   Run Size: 2.00 Mb
   Config Replacement: strict LRU,   Run Replacement: strict LRU
IO Size  Wash Size Config Size  Run Size     APF Percent
-------- --------- ------------ ------------ -----------
2 Kb     512 Kb    0.00 Mb      1.00 Mb      10
16 Kb    192 Kb    1.00 Mb      1.00 Mb      10
(return status = 0)
```

See the *Reference Manual: Procedures*.

Disk Space for dbccdb

Additional disk storage is required for the `dbccdb` database. Because **dbcc checkstorage** uses `dbccdb` extensively, place `dbccdb` on a device that is separate from other database devices.

Note: Do not create `dbccdb` on the master device. Make sure that the log devices and data devices for `dbccdb` are separate.

Segments for Workspaces

You can control workspace placement and improve **dbcc checkstorage** performance by using dedicating segments for workspaces.

When you dedicate new segments for the exclusive use of workspaces, use **sp_dropsegment** to unmap the devices attached to these segments from the default segment.

Creating the dbccdb Database

By default, the `dbccdb` database is not installed by default; you must create it for dbcc checkstorage can record configuration information for target databases, operation activity, and the results of the operations.

1. Run **sp_plan_dbccdb** in the `master` database to obtain recommendations for database size, devices, workspace sizes, cache size, and the number of worker processes for the target database.
2. If `dbccdb` already exists, drop it and all associated devices before creating a new `dbccdb` database:

```
use master
go
if exists (select * from master.dbo.sysdatabases
     where name = "dbccdb")
begin
     print "+++ Dropping the dbccdb database"
     drop database dbccdb
end
go
```

3. Initialize disk devices for the `dbccdb` data and the log:

```
use master
go
disk init
   name = "dbccdb_dat",
   physname = "/remote/disks/masters/",
   size = "4096"
go
disk init
   name = "dbccdb_log",
   physname = "/remote/disks/masters/",
size = "1024"
go
```

4. Create `dbccdb` on the data disk device that you initialized in step 3:

```
use master
go
```

```
create database dbccdb
   on dbccdb_dat = 6
   log on dbccdb_log = 2
go
```

5. (Optional) Add segments for the `scan` and `text` workspaces to the `dbccdb` data device:

```
use dbccdb
go
sp_addsegment scanseg, dbccdb, dbccdb_dat
go
sp_addsegment textseg, dbccdb, dbccdb_dat
go
```

6. Create the tables for `dbccdb` and initialize the `dbcc_types` table:

   ```
   isql -Ujms -P***** -iinstalldbccdb
   ```

   installdbccdb checks for the existence of the database before it attempts to create the tables. It creates only those tables that do not already exist in `dbccdb`. If any of the `dbccdb` tables become corrupted, remove them with **drop table**, and then use **installdbccdb** to re-create them.

7. Create and initialize the `scan` and `text` workspaces:

   ```
   use dbccdb
   go
   sp_dbcc_createws dbccdb, scanseg, scan_pubs2, scan, "64K"
   sp_dbcc_createws dbccdb, textseg, text_pubs2, text, "64K"
   ```

When you have finished installing `dbccdb`, you must update the **dbcc_config** table.

Updating the dbcc_config Table

The **dbcc_config** table describes the currently executing or last completed **dbcc checkstorage** operation.

This includes:

- The location of resources dedicated to the **dbcc checkstorage** operation
- Resource usage limits for the **dbcc checkstorage** operation

This section describes how to update the values in this table.

Viewing the Current Configuration Values

The **sp_dbcc_configreport** *defaults* parameter allows you to view the configured default values.

The *defaults* parameter is optional, and is ignored if *dbname* is not null. Valid values for the *defaults* parameter are true or false (the default). A value of true indicates that you want to display only the default on the configured values. A value of false indicates that you want to view all configured values.

For example to view only the configured default values, use either:

```
sp_dbcc_configreport null, 'true'
```

To view all configured values, do not provide a value for the defaults parameter, or use “false:”

```
sp_dbcc_configreport
```

or:

```
sp_dbcc_configreport null, 'false'
```

Default Configuration Values

sp_dbcc_updateconfig allows you to provide default configuration values for **dbcc** configuration parameters.

You can provide individual configuration values for every database that **checkstorage** analyzed, or you can set default values that are applicable to all databases that do not have a corresponding configuration value (that is, configuration values cannot conflict).

Deleting Configuration Values

sp_dbcc_updateconfig accepts **delete** as a value for the *str1* parameter, which allows you to delete configuration values that you have set on databases.
For example, to delete the default value of the **max worker processes** configuration parameter, enter:

```
sp_dbcc_updateconfig null, 'max worker processes', 'delete'
```

To delete the value of the **max worker processes** configuration parameter for the database `my_db`, enter:

```
sp_dbcc_updateconfig my_db, 'max worker processes', 'delete'
```

dbccdb Maintenance Tasks

You may occasionally need to perform maintenance tasks on `dbccdb`.

- Reevaluate and update the configuration using:
  - **sp_dbcc_evaluatedb** – recommends values for configuration parameters using the results of previous **dbcc checkstorage** operations.
  - **sp_dbcc_updateconfig** – updates the configuration parameters for the specified database.
- Clean up obsolete data:
  - **sp_dbcc_deletedb** – deletes all the information on the specified database from `dbccdb`.
  - **sp_dbcc_deletehistory** – deletes the results of the **dbcc checkstorage** operations on the specified database from `dbccdb`.
- Remove unnecessary workspaces.
- Perform consistency checks on `dbccdb` itself.

Reevaluating and Updating the dbccdb Configuration

If the characteristics of user databases change, use **sp_dbcc_evaluatedb** to reevaluate the current `dbccdb` configuration and recommend more suitable values.

The following changes to user databases might affect the `dbccdb` configuration, as follows:

- When a user database is created, deleted or altered, the size of the workspaces and named cache, or the number of worker threads stored in the dbcc_config table might be affected.
- Changes in the named cache size or worker process count for **dbcc_checkstorage** may require you to reconfigure buffer cache and worker processes.

If the results of **dbcc checkstorage** operations are available for the target database, use **sp_dbcc_evaluatedb** to determine new configuration values. **sp_dbcc_configreport** also reports the configuration parameters for the specified database.

Use **sp_dbcc_updateconfig** to add new databases to the **dbcc_config** table and to change the configuration values in dbcc_config to reflect the values recommended by **sp_dbcc_evaluatedb**.

Cleaning Up dbccdb

SAP ASE stores data generated by **dbcc checkstorage** in dbccdb. Periodically clean up dbccdb by using **sp_dbcc_deletehistory** to delete data from a database. You must have created the database before the date specified.

When you delete a database, you should also delete from dbccdb all configuration information and **dbcc checkstorage** results related to that database. Use **sp_dbcc_deletedb** to delete all database information from dbccdb.

To remove unnecessary workspaces, in dbccdb, issue:

```
drop table workspace_name
```

Performing Consistency Checks on dbccdb

Limited update activity in the dbccdb tables should make corruption infrequent; however, severe corruption might cause **dbcc checkstorage** to fail.

Two signs of corruption in dbccdb are:

- Failure of **dbcc checkstorage** during the initialization phase, as it evaluates the work that needs to be performed, or during the completion phase, when it records its results
- Loss of information about faults resulting from corruption in the recorded faults, found by **dbcc checkstorage**

For **dbcc checkstorage** to locate severe corruptions in dbccdb, you can create an alternate database, dbccalt, which you use only for checking dbccdb. Create dbccalt using the same process that you used to create dbccdb as described in :

If no free devices are available for dbccalt, you can use any device that is not used by the master database or dbccdb.

dbcc checkstorage and the **dbcc** system procedures function the same with dbccalt as they do with dbccdb. When the target database is dbccdb, **dbcc checkstorage** uses dbccalt, if it exists. If dbccalt does not exist, dbccdb can be checked using itself as the

management database. If the target database is dbccdb and dbccalt exists, the results of **dbcc checkstorage** operations on dbccdb are stored in dbccalt. If dbccalt does not exist, the results are stored in dbccdb itself.

Alternatively, you can use **dbcc checkalloc** and **dbcc checktable** to check dbccdb.

If dbccdb becomes corrupted, you can drop it and re-create it, or load an older version from a backup. If you drop it, some of its diagnostic history is lost.

Generating Reports from dbccdb

dbccdb includes several stored procedures so that you can generate reports from the data in dbccdb.

These **dbcc** system procedures report on configuration and statistical information:

- **sp_dbcc_summaryreport**
- **sp_dbcc_configreport**
- **sp_dbcc_statisticsreport**
- **sp_dbcc_faultreport short**

sp_dbcc_fullreport runs a full report on all these system procedures.

See the *Commands Reference Manual: Procedures*.

Reporting a Summary of dbcc checkstorage Operations

sp_dbcc_summaryreport reports all **dbcc checkstorage** operations that were completed for the specified database on or before the specified date.

The following example shows **sp_dbcc_summaryreport** output:

```
sp_dbcc_summaryreport
```

```
DBCC Operation : checkstorage
Database Name      Start time           End Time   Operation ID
         Hard Faults Soft Faults Text Columns Abort Count
         User Name
------------------ -------------------- ---------- ------------
         ---------- ---------- ------------ -----------
         ------------------------------
sybsystemprocs     05/12/1997 10:54:45  10:54:53              1
         0           0            0           0
     sa
sybsystemprocs     05/12/1997 11:14:10  11:14:19              2
         0           0            0           0
     sa
```

See *Reference Manual: Procedures > dbcc Stored Procedures*.

Upgrading Compiled Objects with dbcc upgrade_object

SAP ASE upgrades compiled objects based on their source text.

Compiled objects are:

- Check constraints
- Defaults
- Rules
- Stored procedures (including extended stored procedures)
- Triggers
- Views

The source text of each compiled object is stored in the `syscomments` table, unless it has been manually deleted. When you upgrade the server, the existence of the source text in `syscomments` is verified during that process. However, the compiled objects are not actually upgraded until they are invoked.

For example, if you have a user-defined stored procedure named **list_proc**, the presence of source text for **list_proc** is verified when you upgrade to the latest version of SAP ASE. The first time **list_proc** is invoked after the upgrade, SAP ASE detects that the **list_proc** compiled object has not been upgraded. SAP ASE recompiles **list_proc**, based on the source text in `syscomments`. The newly compiled object is then executed.

Upgraded objects retain the same object ID and permissions that they used before being upgraded.

Compiled objects for which the source text was hidden using **sp_hidetext** are upgraded in the same manner as objects for which the source text is not hidden. See the *Reference Manual: Procedures*.

Note: If you are upgrading from 32-bit installations to use a 64-bit SAP ASE, the size of each 64-bit compiled object in the `sysprocedures` table in each database increases by approximately 55 percent when the object is upgraded. The pre-upgrade process calculates the exact size. Increase your upgraded database size accordingly.

To ensure that compiled objects have been successfully upgraded before they are invoked, you can upgrade them manually using the **dbcc upgrade_object** command.

Finding Compiled Object Errors Before Production

Changes made in versions of SAP ASE earlier than 12.5.x may cause compiled objects to work differently once you upgrade to a later version.

Use **dbcc upgrade_object** to find the errors and potential problem areas that may require manual changes to achieve the correct behavior.

After reviewing the errors and potential problem areas, and fixing those that need to be changed, use **dbcc upgrade_object** to upgrade compiled objects manually instead of waiting for the server to upgrade the objects automatically. See the *Reference Manual: Commands*.

Reserved Word Errors

If **dbcc upgrade_object** finds a reserved word used as an object name in a compiled object, it returns an error, and that object is not upgraded.

To fix the error, either manually change the object name or use quotes around the object name, and issue the command **set quoted identifiers on**. Then, drop and re-create the compiled object.

For example, suppose you load a database dump from an older verion of SAP ASE into a current version, and the dump contains a stored procedure that uses the word "XYZ," which is a reservered word in the current version. When you run **dbcc upgrade_object** on that stored procedure, the command returns an error because, although "XYZ" was not reserved in the older version, it became a reserved word in the current version. With this advance notice, you can change the stored procedure and any related tables before they are used in a production environment.

Missing, Truncated, or Corrupted Source Text

If the source text in `syscomments` was deleted, truncated, or otherwise corrupted, **dbcc upgrade_object** may report syntax errors.

If the source text was not hidden, use **sp_helptext** to verify the completeness of the source text. If truncation or other corruption has occurred, drop and re-create the compiled object.

Temporary Table References

If a compiled object such as a stored procedure or trigger refers to a temporary table (#temp *table_name*) that was created outside the body of the object, the upgrade fails, and **dbcc upgrade_object** returns an error.

To correct this error, create the temporary table exactly as expected by the compiled object, then execute **dbcc upgrade_object** again. You need not do this if the compiled object is upgraded automatically when it is invoked.

Resolving select * Potential Problem Areas

If **dbcc upgrade_object** finds a **select *** clause in the outermost query block of a stored procedure, it returns an error, and does not upgrade the object.

For example, consider the following stored procedures:

```
create procedure myproc as
  select * from employees
go
```

```
create procedure yourproc as
  if exists (select * from employees)
```

```
    print "Found one!"
go
```

dbcc upgrade_object returns an error on **myproc** because **myproc** includes a statement with a **select *** clause in the outermost query block. This procedure is not upgraded.

dbcc upgrade_object does not return an error on **yourproc** because the **select *** clause occurs in a subquery. This procedure is upgraded.

If **dbcc upgrade_object** reports the existence of **select *** in a view, compare the output of `syscolumns` for the original view to the output of the table, to determine whether columns have been added to or deleted from the table since the view was created.

For example, suppose you have the following statement:

```
create view all_emps as select * from employees
```

Before upgrading the `all_emps` view, use the following queries to determine the number of columns in the original view and the number of columns in the updated table:

```
select name from syscolumns
  where id = object_id("all_emps")
select name from syscolumns
  where id = object_id("employees")
```

Compare the output of the two queries. If the table contains more columns than the view, and retaining the pre-upgrade results of the **select *** statement is important, change the **select *** statement to a **select** statement with specific column names. If the view was created from multiple tables, check the columns in all tables that comprise the view and rewrite the **select** statement if necessary.

Warning! Do not execute a **select *** statement from the view. Doing so upgrades the view and overwrites the information about the original column information in `syscolumns`.

Another way to determine the difference between the columns in the view and in the new tables is to run **sp_help** on both the view and the tables that comprise the view.

This comparison works only for views, not for other compiled objects. To determine whether **select *** statements in other compiled objects need to be revised, review the source text of each compiled object.

Using Database Dumps in Upgrades

You can load pre-12.5 database dumps and transaction logs and upgrade the databases. You can also chose to upgrade compiled objects as they are used.

However, be aware that:

- Upgrading requires space for copying data and logging changes to the system tables during the upgrade process. If the source database in the dump was nearly full, the upgrade process might fail due to insufficient space. While this is expected to be uncommon, you can use **alter database** to extend the free space in the event of insufficient-space errors.

- After reloading an older dump, run **sp_checkreswords** from the new installation on the loaded database to check for reserved words.

Upgrading Compiled Objects in Database Dumps

When you load a database dump that was created in an earlier version than the current SAP ASE, you are not required to perform the pre-upgrade tasks before loading the dump. Therefore, you do not receive any notification if the compiled objects in your database dump are missing their source text.

After loading a database dump, run **sp_checksource** to verify the existence of the source text for all compiled objects in the database. Then, you can allow the compiled objects to be upgraded as they are executed, or you can run **dbcc upgrade_object** to find potential problems and upgrade objects manually.

See the *Reference Manual: Procedures*.

To determine whether a compiled object has been upgraded:

- Look at the `sysprocedures.version` column. If the object was upgraded, this column contains the number 15000 for any 15.x version of SAP ASE.
- If you are upgrading to a 64-bit pointer size in the same version, look at the `sysprocedures.status` column. If the object has been upgraded, and is using 64-bit pointers, this column contains a hexadecimal bit setting of 0x2 to indicate that the object uses 64-bit pointers. If this bit is not set, it indicates the object is still a 32-bit object, and has not been upgraded.

CHAPTER 13 Developing a Backup and Recovery Plan

Having a well-documented backup and recovery plan is crucial for any SAP ASE site.

See also

- *Moving the Transaction Log to Another Device* on page 122

Tracking Database Changes

SAP ASE uses transactions to keep track of all database changes. Transactions are SAP ASE units of work. A transaction consists of one or more Transact-SQL statements that succeed —or fail—as a unit.

Each SQL statement that modifies data is considered a *transaction*. Users can also define transactions by enclosing a series of statements within a **begin transaction...end transaction** block. See "Transactions: Maintaining Data Consistency and Recovery," in the *Transact-SQL Users Guide*.

Each database has its own *transaction log*, the system table `syslogs`. The transaction log automatically records every transaction issued by each database user. You cannot turn off transaction logging.

The transaction log is a *write-ahead log*. When a user issues a statement that modifies the database, SAP ASE writes the changes to the log. After all changes for a statement have been recorded in the log, they are written to an in-cache copy of the data page. The data page remains in cache until the memory is needed for another database page. At that time, it is written to disk.

If any statement in a transaction fails to complete, SAP ASE reverses all changes made by the transaction. SAP ASE writes an "end transaction" record to the log at the end of each transaction, recording the status (success or failure) of the transaction.

Getting Information About the Transaction Log

The transaction log contains enough information about each transaction to ensure that it can be recovered.

Use the **dump transaction** command to copy the information it contains to tape or disk. Use **sp_spaceused syslogs** to check the size of the log, or **sp_helpsegment logsegment** to check the space available for log growth.

Warning! Do not use **insert**, **update**, or **delete** commands to modify `syslogs`.

Determining When Log Records Are Committed

set delayed_commit is a performance option suitable only for certain applications.

It increases SAP ASE performance for data manipulation language (DML) operation, (for example, **insert**, **update**, **delete**), but increases the risk of losing your data during a system failure. Performance gains depend on the application in use.

The types of applications that benefit from **set delayed_commit** typically include short transactions that are sent rapidly and serially to SAP ASE. For example, a batch application that issues many **insert** statements, with each **insert** being a separate transaction.

Use the **set** command to enable **delayed_commit** for a session, or with **sp_dboption** for the database.

After you enable **set delayed_commit**, the client application is notified of a successful **commit** before the corresponding log records are written to disk. This improves performance because all but the last log page are written to disk, alleviating contention on the last and active log page.

Before you enable **set delayed_commit**, consider:

- Issuing **shudown with nowait** can cause data durability issues unless you issue a **checkpoint** that finishes before the server shuts down.
- Enabling **set delayed_commit** for a session affects only that session. All other sessions' transactions have all their properties enforced, including their ACID properties. This also means other sessions' physical logs write the last log page and the log records corresponding to a session with **set delayed_commit** enabled.
- **set delayed_commit** is redundant on temporary databases and does not provide a performance improvement.
- Use **set delayed_commit** only after careful consideration of both your application and operational requirements and your environment. While the risk to data durability may be very low, the options for recovery may be time-consuming if your database is large and your tolerance for missing data is low.

Changes to Logging Behavior

Logging behavior changes when you enable **delayed_commit**.

When a session implicitly or explicitly commits a transaction:

- The user log cache (ULC) is flushed to the transaction log in memory.
- The task issues writes on all non-written log pages except the last (which contains the **commit**).
- The task notifies the client application of a successful **commit** without waiting for the I/O to complete.

  This transaction's "last log page" is written:
  - By another transaction when it is no longer the "last log page."

    - By another, non-delayed transaction when it completes.
    - By a **checkpoint** or the housekeeper buffer wash mechanism.
    - By implicit checkpoints causes (for example, **shutdown**, **dump database**, **dump tran**, **sp_dboption truncate log on checkpoint**).
- The task is ready to continue with the next transaction.

Risks of Using delayed_commit

When you enable **set delayed_commit**, SAP ASE notifies the client application before the actual physical disk write completes.

Because of this, the application perceives that the transaction is complete whether or not the physical disk write is successful. In the event of a system failure (disk errors, system crash, and so on), transactions that were not written to disk (transactions whose **commit** records were on the last log page) are not present after recovery in spite of the application being notified they were committed.

Systems that require tight system interdependencies, such as through a messaging system using Real Time Data Services (RTDS), further complicate the consequences of using **set delayed_commit**.

There are two situations where applications can manage the risk:

- The application maintains its own trace or log, and, after a system failure, ensures that the database state corresponds to its own trace or log.
- You can restore the database to the state it was in before the application was run. This assumes you took a complete database backup before a batch-job type application is run. In case of failure, the database backup is loaded and the batch job is restarted.

Enabling set delayed_commit

You can enable **set delayed_commit** for a database or for a session, with the session setting overruling the database setting. This means that a session that enables the option has **delayed_commit** enabled regardless of the database setting.

Designating Responsibility for Backups

Only a system administrator, a database owner, or an operator can execute the **dump** and **load** commands.

Many organizations have an operator who performs all backup and recovery operations. The database owner can dump only his or her own database. The operator and system administrator can dump and load any database.

Checkpoints: Synchronizing a Database and Its Log

A checkpoint writes all dirty pages—pages that have been modified in memory, but not on disk, since the last checkpoint—to the database device.

The SAP ASE automatic *checkpoint* mechanism guarantees that data pages changed by completed transactions are regularly written from the memory cache to the database device. Synchronizing the database and its transaction log shortens the time it takes to recover the database after a system failure.

Setting the Recovery Interval

Typically, automatic recovery takes anywhere between a few seconds and a few minutes per database.

The amount of time depends on the size of the database, the size of the transaction log, and the number and size of the transactions that must be committed or rolled back.

Use **sp_configure** with the **recovery interval in minutes** parameter to specify the maximum permissible recovery time. SAP ASE runs automatic checkpoints often enough to recover the database within the period of time you specify.

The default value, 5, allows recovery within 5 minutes per database. To change the recovery interval to 3 minutes, use:

```
sp_configure "recovery interval in minutes", 3
```

Note: The recovery interval has no effect on long-running, minimally logged transactions (such as **create index**) that are active at the time SAP ASE fails. It may take as much time to reverse these transactions as it took to run them. To avoid lengthy delays, dump each database immediately after you create an index on one of its tables.

Automatic Checkpoint Procedure

Approximately once a minute, a checkpoint task checks each database on the server to see how many records have been added to the transaction log since the last checkpoint.

If the server estimates that the time required to recover these transactions is greater than the database's recovery interval, SAP ASE issues a checkpoint.

The modified pages are written from cache onto the database devices, and the checkpoint event is recorded in the transaction log. Then, the checkpoint task "sleeps" for another minute.

To see the checkpoint task, execute **sp_who**. The checkpoint task usually appears as "CHECKPOINT SLEEP" in the "cmd" column:

```
fid  spid  status      loginame  origname  hostname         blk_spi
d  dbname
        tempdbname    cmd               block_xloid   threadpool
---  ----  ----------  --------  --------  ----------------  -------
```

```
-  ------
        ----------    -----------------  -----------   -------------
------
. . .
0       2    sleeping      NULL      NULL                NULL
 0  master
           tempdb     DEADLOCK TUNE             0    syb_default_pool
0       3    sleeping      NULL      NULL                NULL
 0  master
           tempdb     ASTC HANDLER              0    syb_default_pool
0       4    sleeping      NULL      NULL                NULL
 0  master
           tempdb     CHECKPOINT SLEEP          0    syb_default_pool
. . .
```

SAP ASE inserts a checkpoint record immediately after upgrading a user database. SAP ASE uses this record to ensure that a **dump database** occurs before a **dump transaction** occurs on the upgraded database.

Truncating the Log After Automatic Checkpoints

System administrators can truncate the transaction log when SAP ASE performs an automatic checkpoint.

To set the **trunc log on chkpt** database option, which truncates the transaction log when an automatic checkpoint occurs, execute this command from the `master` database:

```
sp_dboption database_name, "trunc log on chkpt", true
```

This option is not suitable for production environments because it does not make a copy of the transaction log before truncating it. Use **trunc log on chkpt** only for:

- Databases whose transaction logs cannot be backed up because they are not on a separate segment
- Test databases for which current backups are not important

Note: If you leave the **trunc log on chkpt** option set to **off** (the default condition), the transaction log continues to grow until you truncate it with the **dump transaction** command.

To protect your log from running out of space, design your last-chance threshold procedure to dump the transaction log.

Free Checkpoints

When SAP ASE has no user tasks to process, a housekeeper wash task automatically begins writing dirty buffers to disk. Checkpoints that occur as a result of the housekeeper wash task are known as free checkpoints.

If the housekeeper task can flush all active buffer pools in all configured caches, it wakes up the checkpoint task. The checkpoint task determines whether it must perform a checkpoint on the database.

Free checkpoints do not involve writing many dirty pages to the database device, since the housekeeper wash task has already done this work. They may result in a shorter recovery time for the database.

See *Performance and Tuning Series: Basics > Using Engines and CPUs*.

Manually Requesting a Checkpoint

Database owners can issue the **checkpoint** command to force all modified pages that are in memory to be written to disk.

Manual checkpoints do not truncate the log, even if the **trunc log on chkpt** option of **sp_dboption** is turned on.

Use the **checkpoint** command:

- As a precautionary measure in special circumstances—for example, just before a planned **shutdown with nowait** so that SAP ASE recovery mechanisms occur within the recovery interval. (An ordinary **shutdown** performs a checkpoint.)
- To cause a change in database options to take effect after executing **sp_dboption**. (After you run **sp_dboption**, an informational message reminds you to run **checkpoint**.)

You can use **checkpoint** to identify one or more databases or use an **all** clause, which checkpoints all databases. See the *Reference Manual: Commands*.

Automatic Recovery After a System Failure or Shutdown

Each time you restart SAP ASE—for example, after a power failure, an operating system failure, or the use of the **shutdown** command—it automatically performs a set of recovery procedures on each database.

The recovery mechanism compares each database to its transaction log. If the log record for a particular change is more recent than the data page, the recovery mechanism reapplies the change from the transaction log. If a transaction was ongoing at the time of the failure, the recovery mechanism reverses all changes that were made by the transaction.

1. Recovers `master`.
2. Recovers `sybsystemprocs`.
3. Recovers `model`.
4. Creates `tempdb` (by copying `model`).
5. Recovers `sybsystemdb`.
6. Recovers `sybsecurity`.
7. Recovers user databases, in order by `sysdatabases.dbid`, or according to the order specified by **sp_dbrecovery_order**.

Users can log in to SAP ASE as soon as the system databases have been recovered, but they cannot access other databases until they have been recovered.

The configuration variable **print recovery information** determines whether SAP ASE displays detailed messages about each transaction on the console screen during recovery. By default, these messages do not appear. To display messages, use:

```
sp_configure "print recovery information", 1
```

Fast Recovery

During a server restart after a planned or unplanned shutdown, or during high availability failover, a significant portion of time is spent on database recovery. Faster recovery minimizes database downtime.

The goal of fast recovery is to:

- Enhance the performance of database recovery
- Recover multiple databases in parallel by making use of available server resources and tuning them intelligently
- Provide multiple checkpoint tasks at runtime that can run concurrently to minimize the work at recovery time

SAP ASE Start-up Sequence

SAP ASE performes a specific sequence of steps at start-up.

1. System databases are recovered on engine 0.
2. SAP ASE accepts user connections.
3. All engines that are configured to be online during start-up are brought online.
4. User databases are recovered in parallel by a “self-tuned” number of recovery tasks using the default data cache tuned for optimal recovery performance.

During a fail over in a high availability system, failed over user databases are recovered and brought online in parallel.

Bringing Engines Back Online

Engines are brought online after system databases are recovered, and before user databases, allowing user databases to be recovered in parallel, and makes the engines available for online activities.

Engines are brought online in this fashion only during start-up. In all other circumstances, such as failover, engines are already online on the secondary server.

Parallel Recovery

Databases are recovered in parallel by multiple recovery tasks during start-up and failover.

Database recovery is an I/O-intensive process. The time to recover SAP ASE with parallel recovery depends on the bandwidth of the underlying I/O subsystem. The I/O subsystem should be able to handle SAP ASE concurrent I/O requests.

With parallel recovery, multiple tasks recover user databases concurrently. The number of recovery tasks is dependent on the configuration parameter **max concurrently recovered db**. The default value of 0 indicates that SAP ASE adopts a self-tuning approach in which it does not make any assumptions on the underlying storage architecture. Statistical I/O sampling methods determine the optimal number of recovery tasks depending on the capabilities of the underlying I/O subsystem. An advisory on the optimal number of recovery tasks is provided. If the configuration value is nonzero, SAP ASE spawns as many tasks as indicated by the configuration parameter and also by the **number of open databases** parameter.

During parallel recovery, the system administrator can force serial recovery by setting **max concurrently recovered db** to 1. The active recovery tasks drain out after completing the recovery of the database that is being worked on. The remaining databases are recovered serially.

See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Database Recovery

SAP ASE database recovery includes checking buffer pool sizes and setting **async prefetch**.

- Log I/O size – SAP ASE uses the largest buffer pool available in the default data cache for log I/O. If a pool with the largest buffer size is unavailable, the server dynamically creates this pool, and uses the pool for log I/O. The buffers for this pool come from the default pool. Recovery tunes the size of the large buffer pool for optimal recovery performance. If the large pool is available but its size is not optimal, SAP ASE dynamically resizes it, and the default pool, for optimal recovery performance. The buffer pool configurations are restored at the end of recovery.

  See *Performance and Tuning Series: Basics > Memory Use and Performance*.
- **async prefetch limit** – during recovery, the server automatically sets the local **async prefetch** limit for the pools in the default data cache used by recovery to an optimal value. This overrides any user specifications for the duration of recovery.

  When recovery completes, the original configuration values are restored.

Specifying the Recovery Order

Users can specify the order in which databases are recovered, for all or a subset of user databases.

You can use **sp_dbrecovery_order** to configure more important databases to be recovered earlier. You must be in the `master` database and have system administrator privileges to use **sp_dbrecovery_order** to enter or modify a user-defined recovery order. Any user, in any database, can use **sp_dbrecovery_order** to the user-defined recovery order of databases. See *Reference Manual: Procedures*.

sp_dbrecovery_order has an additional parameter indicating the online ordering.

```
sp_dbrecovery_order [database_name [, rec_order [, force [ relax |
strict ]]]]
```

- **relax** – the databases are made as they recover (default).
- **strict** – the databases are specified by the recovery order.

The default is **relax**, which means that databases are brought online immediately when recovery has completed.

Recovery order must be consecutive, starting with 1. You cannot assign a recovery sequence of 1, 2, 4, with the intention of assigning a recovery order of 3 to another database at a later time.

To insert a database into a user-defined recovery sequence without putting it at the end, enter *rec_order* and specify **force**. For example, if databases `A`, `B`, and `C` have a user-defined recovery order of 1, 2, 3, and you want to insert the `pubs2` database as the second user database to recover, enter:

```
sp_dbrecovery_order pubs2, 2, force
```

This command assigns a recovery order of 3 to database B and a recovery order of 4 to database C.

SAP ASE 12.5.1 and later uses parallel recovery tasks to determine the next database to recover according to the user-specified order. The remaining databases are recovered in the order of their database IDs. The time to recover a database is dependent on many factors, including the size of the recoverable log. Therefore, although you determined the recovery order with **sp_dbrecovery_order**, SAP ASE may complete the database recovery in an order other than which it started. For applications that must enforce that databases are brought online in the same order as the recovery order, SAP ASE provides the **strict** option in **sp_dbrecovery_order**.

Changing or Deleting the Recovery Position of a Database

To change the position of a database in a user-defined recovery sequence, delete the database from the recovery sequence, then insert it in the position you want it to occupy.

If the new position is not at the end of the recovery order, use the **force** option.

To delete a database from a recovery sequence, specify a recovery order of -1.

For example, to move the `pubs2` database from recovery position 2 to recovery position 1, delete the database from the recovery sequence and then reassign it a recovery order as follows:

```
sp_dbrecovery_order pubs2, -1
sp_dbrecovery_order pubs2, 1, "force"
```

Listing the User-Assigned Recovery Order of Databases

Use **sp_dbrecovery_order** to list the recovery order of all databases that are assigned a recovery order.

sp_dbrecovery_order generates output similar to:

```
The following databases have user specified recovery order:
Recovery Order Database Name        Database Id
-------------- -------------------- -----------
    1          dbccdb                       8
    2          pubs2                        5
    3          pubs3                        6
    4          pubtune                      7
The rest of the databases will be recovered in default database id
order.
```

To display the recovery order of a specific database, enter the database name:

```
1> sp_dbrecovery_order pubs2
2> go
```

```
Database Name Database id Recovery Order
 ------------- ----------- --------------
 pubs2             5           2
```

Parallel Checkpoints

A pool of checkpoint tasks works on the list of active databases in parallel.

This pool is controlled by the configuration parameter **number of checkpoint tasks**. Where there is a checkpoint bottleneck, more checkpoint tasks translate to shorter recoverable logs, and recovery has less work to do in case of a failure, thus improving availability.

The default value of **number of checkpoint tasks** is 1 for serial checkpoints. The **number of engines** and **number of open databases** limit the value for this parameter. To facilitate parallel recovery, configure the maximum number of engines to be online at start-up. When you reduce the value for this parameter, checkpoints drain out, and when you increase the value, additional tasks are spawned.

Checkpoints are I/O-intensive; therefore, the effectiveness of parallel checkpoints is dependent on the layout of the databases and performance of the underlying I/O subsystem. Tune **number of checkpoint tasks** depending on the number of active databases and the ability of the I/O subsystem to handle writes.

See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Recovery State

The global variable *@@ recovery_state* determines if SAP ASE is in recovery.

The values that *@@ recovery_state* can have are:

- NOT_IN_RECOVERY – SAP ASE is not in start-up recovery or in failover recovery. Recovery has been completed and all databases that can be online are brought online.
- RECOVERY_TUNING – SAP ASE is in recovery (either start-up or failover) and is tuning the optimal number of recovery tasks.
- BOOTIME_RECOVERY – SAP ASE is in start-up recovery and has completed tuning the optimal number of tasks. Not all databases have been recovered.
- FAILOVER_RECOVERY – SAP ASE is in recovery during an HA failover and has completed tuning the optimal number of recovery tasks. All databases are not brought online yet.

@@recovery_state can be used by applications to determine when all the databases are recovered and brought online.

Tuning for Fast Recovery

Review the database layout, runtime configuration, and the data space accounting setting to reduce recovery time.

- Database layout – Databases should have logs and data on their own physical devices. The access patterns for log and data are different and should be kept separate. Configure the underlying I/O subsystem to handle concurrent I/O requests from multiple databases in SAP ASE.
- Runtime configuration – Configure an optimal housekeeper wash percentage controlled by **housekeeper free write percent**, so that during free cycles dirty pages are written out. The default value is usually optimal.

  Ensure that long-running transactions are kept to a minimum. Long- running transactions hold resources and can also cause longer recovery times. To avoid longer recovery times, using **polite shutdown** to shut down the server.
- Setting space accounting – If data space accounting is not essential for a database, set the database option to turn off free space accounting using **sp_dboption**. This disables threshold actions on the data segment

The sybdumptran Utility

Use the **sybdumptran** utility to dump the most recent transactions when, due to the server suffering a catastrophic failure, you cannot use **dump tran with no_truncate**.

Using **dump transaction with no_truncate** allows you to dump the last transactions and recover your database. The **dump transaction with no_truncate** command makes minimal

use of the database when creating a transaction log dump. This ensures that no transactions are lost, allowing you to restore your database to its most recent point in time.

However, **dump transaction with no_truncate** does not work, when the failure is related to an incomplete SAP ASE installation directory or a missing or corrupt master device, as **dump transaction with no_truncate** requires a working SAP ASE in which the `master` database contains valid metadata from the database that failed.

When this type of a catastrophic failure occurs, use **sybdumptran**. This standalone utility generates a transaction log dump from the log pages contained in operating system files or raw devices, that were formerly used by the log segment of a database in an SAP ASE environment. The log devices are located based on information stored in a recent database or transaction log dump, or another metadata file that **sybdumptran** can generate.

The **sybdumptran** utility allows you to obtain a transaction log dump of the as-yet-undumped portion of the transaction log in the case of an SAP ASE catastrophic failure. **sybdumptran**, is unaffected by your inability to access parts of SAP ASE, such as `sysdatabases`, `sysdevices`, and `sysusages`, which **dump tran with no_truncate** needs for information on where the log starts. With **sybdumptran**, even if such information is destroyed or unavailable, you can still obtain a transaction log, as long as the log devices are available.

The syntax for **sybdumptran** is:

```
sybdumptran [options] -o output_file
```

Fault Isolation During Recovery

The recovery procedures, known simply as "recovery," rebuild the server's databases from the transaction logs.

The following situations cause recovery to run:

- SAP ASE start-up
- Use of the **load database** command
- Use of the **load transaction** command

The recovery isolation mode setting controls how recovery behaves when it detects corrupt data while reversing or reapplying a transaction in a database.

If an index is marked as suspect, the system administrator can repair this by dropping and re-creating the index.

Recovery fault isolation provides the ability to:

- Configure whether an entire database, or only the suspect pages, become inaccessible when recovery detects corruption
- Configure whether an entire database with suspect pages comes online in **read_only** mode, or whether only the online pages are accessible for modification

- List databases that have suspect pages
- List the suspect pages in a specified database by page ID, index ID, and object name
- Bring suspect pages online for the system administrator while they are being repaired
- Bring suspect pages online for all database users after they have been repaired

The ability to isolate only the suspect pages while bringing the rest of the database online provides a greater degree of flexibility in dealing with data corruption. You can diagnose problems, and sometimes correct them, while most of the database remains accessible to users. You can assess the extent of the damage and schedule emergency repairs or reload for a convenient time.

Recovery fault isolation applies only to user databases. Recovery always takes a system database entirely offline if it has any corrupt pages. You cannot recover a system database until you have repaired or removed all of its corrupt pages.

Persistence of Offline Pages

Suspect pages that you have taken offline remain offline when you restart the server. Information about offline pages is stored in `master.dbo.sysattributes`.

Use the **drop database** and **load database** commands to clear entries for suspect pages from `master.dbo.sysattributes`.

Configuring Recovery Fault Isolation

When SAP ASE is installed, the default recovery isolation mode is **databases**, which marks a database as suspect and takes the entire database offline if it detects any corrupt pages.

Isolating Suspect Pages

To isolate the suspect pages so that only they are taken offline—while the rest of the database remains accessible to users—use **sp_setsuspect_granularity** to set the recovery isolation mode to **page**.

This mode is in effect the next time that recovery is performed in the database. See the *Reference Manual: Procedures*.

Without the **database** or **page** argument, **sp_setsuspect_granularity** displays the current and configured recovery isolation mode settings for the specified database. Without any arguments, it displays those settings for the current database.

If corruption cannot be isolated to a specific page, recovery marks the entire database as suspect, even if the recovery isolation mode is set to **page**. For example, a corrupt transaction log or the unavailability of a global resource causes this to occur.

When recovery marks specific pages as suspect, the default behavior is for the database to be accessible for reading and writing with the suspect pages offline and therefore inaccessible. However, if you specify the **read_only** option to **sp_setsuspect_granularity**, and recovery marks any pages as suspect, the entire database comes online in **read_only** mode and cannot be modified.

In this case, the suspect pages remain offline until you repair them or force them.

If you prefer the **read_only** option, but in certain cases you are comfortable allowing users to modify non-suspect pages, you can make the online portion of the database writable with **sp_dboption**

```
sp_dboption pubs2, "read only", false
```

Raising the Number of Suspect Pages Allowed

The suspect escalation threshold is the number of suspect pages at which recovery marks an entire database suspect, even if the recovery isolation mode is **page**.

By default, the suspect escalation threshold is set to 20 pages in a single database. Use **sp_setsuspect_threshold** to change the suspect escalation threshold. See the *Reference Manual: Procedures*

You configure recovery fault isolation and the suspect escalation threshold at the database level.

This example shows that the recovery isolation mode for the `pubs2` database was **page** and the escalation threshold was 20 the last time recovery ran on this database (the current suspect threshold values). The next time recovery runs on this database, the recovery isolation mode is **page** and the escalation threshold is 30 (the configured values):

```
sp_setsuspect_granularity pubs2
```

```
DB Name  Cur. Suspect Gran.  Cfg. Suspect Gran.  Online mode
-------  ------------------  ------------------  -----------
pubs2    page                page                read/write
```

```
sp_setsuspect_threshold pubs2
```

```
DB Name       Cur. Suspect threshold   Cfg. Suspect threshold
------------- ------------------------ ----------------------
pubs2         20                       30
```

With no arguments, **sp_setsuspect_granularity** and **sp_setsuspect_threshold** display the current and configured settings for the current database, if it is a user database.

Getting Information About Offline Databases and Pages

Use **sp_listsuspect_db** to see which databases have offline pages.
The following example displays general information about the suspect pages:

```
sp_listsuspect_db
The database 'dbt1' has 3 suspect pages belonging to 2 objects.
```

Use **sp_listsuspect_page** to display detailed information about individual offline pages.

If you do not specify the `dbname`, the default is the current database.

The following example shows the detailed page-level output of **sp_listsuspect_page** in the `dbt1` database

```
sp_listsuspect_page dbt1
```

```
DBName   Pageid       Object    Index   Access
-------- ------------ --------- ------- -------------------
dbt1     384          tab1      0       BLOCK_ALL
dbt1     390          tab1      0       BLOCK_ALL
dbt1     416          tab1      1       SA_ONLY

(3 rows affected, return status = 0)
```

If the value in the `Access` column is SA_ONLY, and the suspect page is 1, the suspect page is accessible only to users with the **sa_role**. If it is BLOCK_ALL, no one can access the page.

Any user can run **sp_listsuspect_db** and **sp_listsuspect_page** from any database.

Bringing Offline Pages Online

Use **sp_forceonline_db** to make all the offline pages in a database accessible, and use **sp_forceonline_page** to make an individual offline page accessible.

Specify the type of access with both of these procedures.

- **"sa_on"** makes the suspect page or database accessible only to users with the **sa_role**. This is useful for repairing the suspect pages and testing the repairs while the database is up and running, without allowing normal users access to the suspect pages. You can also use it to perform a **dump database** or a **dump transaction with no_log** on a database with suspect pages, which would be prohibited if the pages were offline.
- **"sa_off"** blocks access to all users, including system administrators. This reverses a previous **sp_forceonline_db** or **sp_forceonline_page** with "**sa_on**."
- **"all_users"** brings offline pages online for all users after the pages have been repaired.

  Unlike bringing suspect pages online with **"sa_on"** and then making them offline again with "**sa_off**," when you use **sp_forceonline_page** or **sp_forceonline_db** to bring pages online for "all users," this action cannot be reversed. There is no way to make the online pages offline again.

  Warning! SAP ASE does not perform any checks on pages being brought online. Ensure that pages being brought online have been repaired.

You cannot execute **sp_forceonline_db** or **sp_forceonline_page** inside a transaction.

You must have the **sa_role** and be in the `master` database to execute **sp_forceonline_db** and **sp_forceonline_page**.

See the *Reference Manual: Procedures*.

Index-Level Fault Isolation for Data-Only-Locked Tables

When pages of an index for a data-only-locked table are marked as suspect during recovery, the entire index is taken offline.

These system procedures manage offline indexes:

- **sp_listsuspect_object**

- **sp_forceonline_object**

In most cases, a system administrator uses **sp_forceonline_object** to make a suspect index available only to those with the **sa_role**. If the index is on a user table, you can repair the suspect index by dropping and re-creating the index.

See the *Reference Manual: Procedures*.

Side Effects of Offline Pages

Databases with offline pages include restrictions.

- Transactions that need offline data, either directly or indirectly (for example, because of referential integrity constraints), fail and generate a message.
- You cannot use **dump database** when any part of the database is offline.

  A system administrator can force the offline pages online using **sp_forceonline_db** with **"sa_on"** dump the database, and then use **sp_forceonline_db** with **"sa_off"** after the dump completes.
- You cannot use **dump transaction with no_log** or **dump transaction with truncate_only** if any part of a database is offline.

  A system administrator can force the offline pages online using **sp_forceonline_db** with **"sa_on"**, dump the transaction log using **with no_log**, and then use **sp_forceonline_db** with **"sa_off"** after the dump completes.
- To drop a table or index containing offline pages, you must use a transaction in the `master` database. Otherwise, the drop fails because it must delete entries for the suspect pages from `master.dbo.sysattributes`. The following example drops the object and deletes information about its offline pages from `master.dbo.sysattributes`.

  To drop an index named `authors_au_id_ind`, which contains suspect pages, from the `pubs2` database, drop the index inside a `master` database transaction:

```
use master
go
sp_dboption pubs2, "ddl in tran", true
go
checkpoint pubs2
go
begin transaction
drop index authors.au_id_ind
commit
go
use master
go
sp_dboption pubs2, "ddl in tran", false
go
checkpoint pubs2
go
```

Recovery Strategies Using Recovery Fault Isolation

Strategies for returning a database with suspect pages to a consistent state while users are accessing it include reloading the database, or repairing it.

Both strategies require:

- A clean database dump
- A series of reliable transaction log dumps up to the point at which the database is recovered with suspect pages
- A transaction log dump to a device immediately after the database is recovered to capture changes to the offline pages
- Continuous transaction log dumps to devices while users work in the partially offline database

Reload Strategy

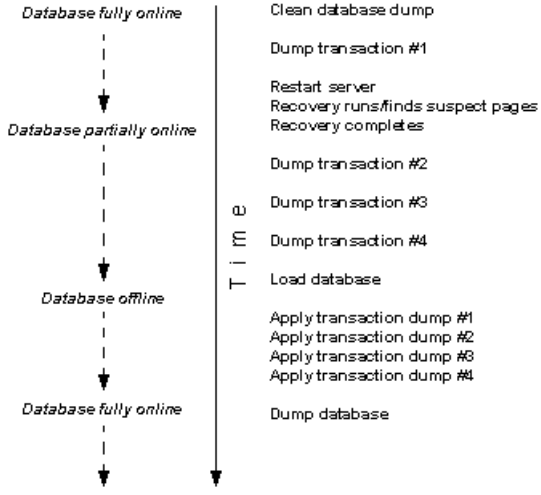
Reloading involves restoring a clean database from backups.

When convenient, load the most recent clean database dump, and apply the transaction logs to restore the database. **load database** clears the suspect page information from the `master.dbo.sysdatabases` and `master.dbo.sysattributes` system tables.

When the restored database is online, dump the database immediately.

This figure illustrates the strategy used to reload databases.

Figure 19: Reload strategy



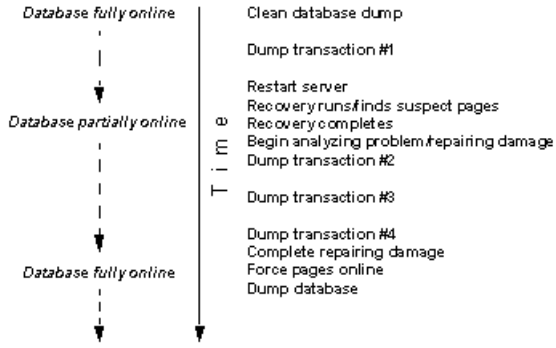
Repair Strategy

The repair strategy involves repairing corrupt pages while the database is partially offline.

Diagnose and repair problems using known methods, including **dbcc** commands, running queries with known results against the suspect pages, and calling SAP Technical Support, if necessary. Repairing damage can also include dropping and re-creating objects that contain suspect pages.

You can either use **sp_forceonline_page** to bring offline pages online individually, as they are repaired, or wait until all the offline pages are repaired and bring them online all at once with **sp_forceonline_db**.

The repair strategy does not require taking the entire database offline. This figure illustrates the strategy used to repair corrupt pages.



Assessing the Extent of Corruption

You can sometimes use recovery fault isolation to assess the extent of corruption by forcing recovery to run and examining the number of pages marked suspect and the objects to which they belong.

For example, if users report problems in a particular database, set the recovery isolation mode to "page," and force recovery by restarting SAP ASE. When recovery completes, use **sp_listsuspect_db** or **sp_listsuspect_page** to determine how many pages are suspect and which database objects are affected.

If the entire database is marked suspect and you receive this message:

```
Reached suspect threshold '%d' for database '%.*s'. Increase suspect
threshold using sp_setsuspect_threshold.
```

Use **sp_setsuspect_threshold** to raise the suspect escalation threshold and force recovery to run again. Each time you get this message, you can raise the threshold and run recovery until the database comes online. If you do not get this message, the corruption is not isolated to

specific pages, in which case this strategy for determining the number of suspect pages does not work.

Using the dump and load Commands

In case of media failure, such as a disk crash, you can restore your databases if—and only if—you have regular backups of the databases and their transaction logs.

Full recovery depends on the regular use of the **dump database** and **dump transaction** commands to back up databases and the **load database** and **load transaction** commands to restore them.

Warning! Never use operating system copy commands to copy an operating database device. Running SAP ASE against a copied device may cause database corruption. Either shutdown SAP ASE, or use the **quiesce database** command to protect copy operations.

The dump commands can complete successfully even if your database is corrupt. Before you back up a database, use the **dbcc** commands to check its consistency.

Warning! If you dump directly to tape, do not store any other types of files (UNIX backups, tar files, and so on) on that tape. Doing so can invalidate the SAP dump files. However, if you dump to a UNIX file system, you can then archive the resulting files to a tape.

See also

- *Shrinking Log Space* on page 129

dump database: Making Routine Database Dumps

The **dump database** command makes a copy of the entire database, including both the data and the transaction log. **dump database** does not truncate the log.

dump database allows *dynamic dumps*, which means that users can continue to make changes to the database while the dump takes place. This makes it convenient to back up databases on a regular basis.

dump transaction: Making Routine Transaction Log Dumps

Use the **dump transaction** command to make routine backups of your transaction log.

dump transaction is similar to the incremental backups provided by many operating systems. It copies the transaction log, providing a record of any database changes made since the last transaction log dump. After **dump transaction** has copied the log, it truncates the inactive portion.

dump transaction takes less time and storage space than a full database backup, and it is usually run more often. Users can continue to make changes to the database while the dump is taking place. You can run **dump transaction** only if the database stores its log on a separate segment.

After a media failure, use the **with no_truncate** option of **dump transaction** to back up your transaction log. This provides a record of the transaction log up to the time of the failure.

dump tran with no_truncate: Copying the Log After Device Failure

If your data device fails and the database is inaccessible, use the **with no_truncate** option of **dump transaction** to get a current copy of the log.

This option does not truncate the log. You can use it only if the transaction log is on a separate segment and the `master` database is accessible.

load database: Restoring the Entire Database

Use the **load database** command to load a backup that was created with **dump database**.

You can load the dump into a preexisting database or create a new database with the **for load** option. When you create a new database, allocate at least as much space as was allocated to the original database.

The **load database** command sets the database status to "offline." This means you do not have to use the **no chkpt on recovery**, **dbo use only**, and **read only** options of **sp_dboption** before you load a database. However, no one can use a database during the database load and subsequent transaction log loads. To make the database accessible to users, issue the **online database** command.

After the database is loaded, SAP ASE may need to:

- "Zero" all unused pages, if the database being loaded into is larger than the dumped database.
- Complete recovery, applying transaction log changes to the data.

Depending on the number of unallocated pages or long transactions, this can take a few seconds, or many hours for a very large database. SAP ASE issues messages that it is zeroing pages or has begun recovery. These messages are normally buffered; to see them, issue:

```
set flushmessage on
```

load transaction: Applying Changes to the Database

After you have loaded the database, use the **load transaction** command to load each transaction log dump in the order in which it was made.

This process reconstructs the database by reexecuting the changes recorded in the transaction log. If necessary, you can recover a database by rolling it forward to a particular time in its transaction log, using the **until_time** option of **load transaction**.

Users cannot make changes to the database between the **load database** and **load transaction** commands, due to the "offline" status set by **load database**.

You can load only transaction log dumps that are at the same release level as the associated database.

When the entire sequence of transaction log dumps has been loaded, the database reflects all transactions that had committed at the time of the last transaction log dump.

online database: Making the Database Available to Users

When the load sequence completes, change the database status to "online," to make it available to users. A database loaded by **load database** remains inaccessible until you issue the **online database** command.

Before you issue **online database**, be sure you have loaded all required transaction logs.

Dumping and Loading Databases Across Platforms

SAP ASE allows you to dump and load databases across platforms with different endian architecture.

This means you can perform **dump database** and **load database** from either a big-endian platform to a little-endian platform, or from a little-endian platform to a big-endian platform.

In a big-endian system, the most significant byte of storage, such as integer or long, has the lower address. The reverse is true for a little-endian system.

Note: When you perform **dump database** and **load database** across platforms with the same endian architecture, user and system data do not require conversions. There are no limitations on operations when dumping and loading a database.

SAP ASE automatically detects the architecture type of the originating system of the database dump file during a **load database**, then performs the necessary conversions. Loads from older versions, such as 12.5.4, are supported. The dump and load can be from 32-bit to 64-bit platforms, and vice versa.

Platforms supported:

Big-endian	Sun Solaris	IBM AIX	HP-UX on HPPA, HPIA
Little-endian	Linux IA	Windows	Sun Solaris x86

Stored procedures and other compiled objects are recompiled from the SQL text in `syscomments` at the first execution after the **load database** for certain combination platforms.

Dumping a Database

Before you run **dump database** for a cross-platform dump and load, move the database to a transactional quiescent status.

1. Execute **dbcc checkdb** and **dbcc checkalloc** to verify the database runs cleanly.
2. Use **sp_dboption** to place the database in a single-user mode.
3. Use **sp_flushstats** to flush statistics to `systabstats`.

4. Wait for 10 to 30 seconds, depending on the database size and activity.
5. Run **checkpoint** against the database to flush updated pages.
6. Run **dump database**.

Loading a Database

Once you load a database, SAP ASE automatically identifies the endian type on the dump file and performs all necessary conversions while the **load database** and **online database** commands are executing.

After SAP ASE converts the index rows, the order of index rows may be incorrect. SAP ASE marks the following indexes on user tables as suspect indexes during execution of **online database**:

- Nonclustered index on APL table
- Clustered index on DOL table
- Nonclustered index on DOL table

During cross-platform dump and load operations, suspect partitions are handled as follows:

- During the first **online database** command, after you execute **load database** across two platforms with different endian types, the hash partition is marked suspect.
- Any global clustered index on a round-robin partition, which has an internally generated partition condition with a `unichar` or `varchar` partition key, is marked suspect.
- After the database is online, use **sp_post_xpload** to fix the suspect partitions and indexes.

Restrictions for Dumping and Loading Databases and Transactions

Dumping and loading databases and transactions includes restrictions.

- **dump transaction** and **load transaction** are not allowed across platforms.
- **dump database** and **load database** to or from a remote Backup Server are not supported across platforms.
- You cannot load a password-protected dump file across platforms.
- If you perform **dump database** and **load database** for a parsed XML object, you must parse the text again after **load database** has completed.
- You can load dumps only to servers that have the same sort order as the server from which they were dumped. For example, you cannot load a dump from a server that uses a dictionary order, case-sensitive, accent-sensitive sort order to a server that uses a dictionary order, case-insensitive, accent insensitive sort order.
- SAP ASE cannot translate embedded data structures stored as `binary`, `varbinary`, or `image` columns.
- **load database** is not allowed on the `master database` across platforms.
- Stored procedures and other compiled objects are recompiled from the SQL text in `syscomments` at the first execution after the **load database**.

If you do not have permission to recompile from text, then the person who does must recompile from text using **dbcc upgrade_object** to upgrade objects.

Note: If you migrate login records in the `syslogins` system table in the master database from Solaris to Linux, you can use **bcp** with character format. The login password from the Solaris platform is compatible on Linux without a trace flag from this release. For all other combinations and platforms, login records need to be re-created because the passwords are not compatible.

Improving Recovery Prefetch

The look-ahead size for prefetching pages of the recovery scan processes can be optimized using the **recovery prefetch size** configuration parameter.

Boot time, **load database**, and **load tran** recovery can prefetch to-be-recovered data pages from disk into cache so that recovery does not need to wait when attempting to redo or undo log records.

Prefetching pages is done by having an auxiliary process operate ahead of the recovery redo or undo scan reading data (and index) pages from disk into cache. Problems can occur if the prefetch auxiliary process:

- Operates too far ahead of the recovery scan – in this case, the recovery scan might find that the pages are no longer in cache, as they have been aged out.
- Operates too close to the recovery scan – in this case, the recovery scan might block the waiting of prefetch disk reads to complete.

The look-ahead size can be optimized using the **recovery prefetch size** configuration parameter. The look-ahead size can be set to be dynamically determined, or you can specify the exact size of the look-ahead in numbers of log records using the **recovery prefetch size** configuration size parameter.

The settings for **recovery prefetch size** are:

- 0 - dynamic look-ahead is applied. That is, SAP ASE determines the optimal look-ahead size. This is the default.
- value other than 0 - the value specifies the exact size of the look-ahead in numbers of log records. Use this only if you find a particular look-ahead size works more effectively that dynamic look-ahead.

Performance Notes

Index rows are ordered for fast access to a table's data row. Index rows that contain row identifiers are treated as binary to achieve fast access to the user table.

Within the same architecture platform, the order of index rows remains valid, and search order for a selection criteria takes its normal path. However, when index rows are translated across different architectures, the order in which optimization was performed is invalidated, resulting in an invalid index on user tables in a cross-platform dump and load.

A database dump from a different architecture, such as big-endian to little-endian, is loaded, certain indexes are marked as suspect:

- Nonclustered index on APL table
- Clustered index on DOL table
- Nonclustered index on DOL table

To fix indexes on the target system, after loading from a different architecture dump, you can either:

- Drop and re-create all of the indexes, or
- Use **sp_post_xpload**.

In general, it requires planning to re-create indexes on large tables, and it can be a lengthy process.

sp_post_xpload validates indexes, drops invalid indexes, and re-creates dropped indexes in a single command on databases. Because **sp_post_xpload** performs many operations, it can take longer dropping and re-creating indexes. Use **sp_post_xpload** for databases smaller than 10G. For databases larger than 10G, SAP recommends that you drop and re-create indexes.

Moving a Database to Another SAP ASE

You can use **dump database** and **load database** to move a database from one SAP ASE to another.

However, you must ensure that the device allocations on the target SAP ASE match those on the original. Otherwise, system and user-defined segments in the new database will not match those in the original database.

To preserve device allocations when loading a database dump into a new SAP ASE, use the same instructions as for recovering a user database from a failed device.

Also, follow these general guidelines when moving system databases to different devices:

- Before moving the `master` database, always unmirror the master device. If you do not, SAP ASE attempts to use the old mirror device file when you start SAP ASE with the new device.
- When moving the `master` database, use a new device that is the same size as the original to avoid allocation errors in `sysdevices`.
- To move the `sybsecurity` database, place the new database in single-user mode before loading the old data into it.

Upgrading a User Database

You can load dumps into the current version of SAP ASE from any version of SAP ASE that is at version 11.9 and later. The loaded database is not upgraded until you issue **online database**.

1. Use **load database** to load a database dump. **load database** sets the database status to "offline."
2. Use **load transaction** to load, *in order*, all transaction logs generated after the last database dump. Load all transaction logs before going to step 3.
3. Use **online database** to upgrade the database. The **online database** command upgrades the database because its present state is incompatible with the current version of SAP ASE. When the upgrade completes, the database status is set to "online," which makes the database available for public use.
4. Make a dump of the upgraded database. A **dump database** must occur before a **dump transaction** command is permitted.

See the *Reference Manual: Commands*.

Occasionally, performing the steps described above does not apply.

The table below describes when to use the special **with no_log** and **with truncate_only** options instead of the standard **dump transaction** command.

Warning! Use the special **dump transaction** commands *only* as indicated below. In particular, use **dump transaction with no_log** as a last resort and use it only once after **dump transaction with no_truncate** fails. The **dump transaction with no_log** command frees very little space in the transaction log. If you continue to load data after entering **dump transaction with no_log**, the log may fill completely, causing any further **dump transaction** commands to fail. Use **alter database** to allocate additional space to the database.

Table 8. When to use dump transaction with truncate_only or with no_log

When	Use
The log is on the same segment as the data.	**dump transaction with truncate_only** to truncate the log **dump database** to copy the entire database, including the log
You are not concerned with the recovery of recent transactions (for example, in an early development environment).	**dump transaction with truncate_only** to truncate the log **dump database** to copy the entire database

When	Use
Your usual method of dumping the transaction log (either the standard **dump transaction** command or **dump transaction with truncate_only**) fails because of insufficient log space.	**dump transaction with no_log** to truncate the log without recording the event **dump database** immediately afterward to copy the entire database, including the log

See also

- *Other Times to Back Up a Database* on page 306

Using the Special Load Options to Identify Dump Files

Use the **with headeronly** option to provide header information for a specified file or for the first file on a tape.

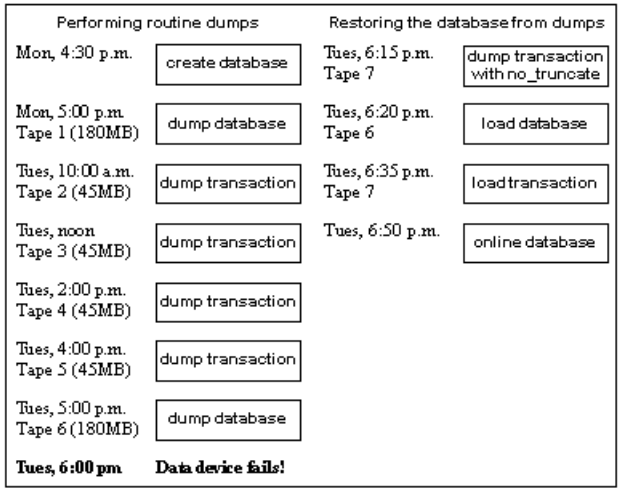
Use the **with listonly** option to return information about all files on a tape. These options do not actually load databases or transaction logs on the tape.

Note: These options are mutually exclusive. If you specify both, **with listonly** takes precedence.

Restoring a Database from Backups

Restoring a database includes a series of **dump** and **load** commands.

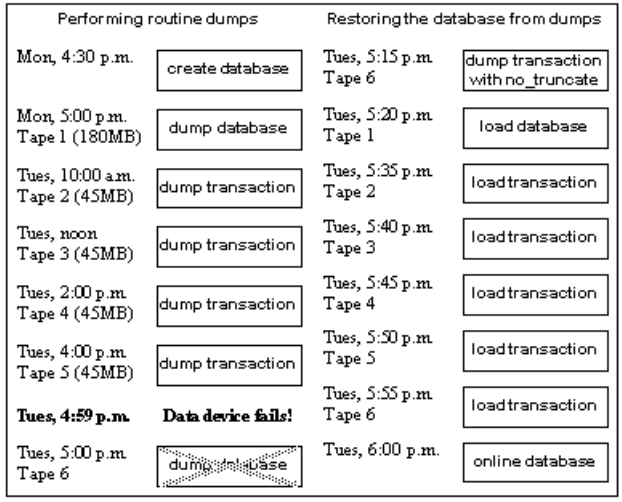
The figure below illustrates the process of restoring a database that is created at 4:30 p.m. on Monday and dumped immediately afterward. Full database dumps are made every night at 5:00 p.m. Transaction log dumps are made at 10:00 a.m., 12:00 p.m., 2:00 p.m., and 4:00 p.m. every day:



1. Use **dump transaction with no_truncate** to get a current transaction log dump
2. Use **load database** to load the most recent database dump, Tape 6. **load database** sets the database status to "offline."
3. Use **load transaction** to apply the most recent transaction log dump, Tape 7.
4. Use **online database** to set the database status to "online."

This figure illustrates how to restore the database when the data device fails at 4:59 p.m. on Tuesday—just before the operator is scheduled to make the nightly database dump:

Figure 20: Restoring a database, a second scenario



To restore the database:

1. Use **dump transaction with no_truncate** to get a current transaction log dump on Tape 6 (the tape you would have used for the routine database dump).
2. Use **load database** to load the most recent database dump, Tape 1. **load database** sets the database status to "offline."
3. Use **load transaction** to load Tapes 2, 3, 4, and 5 and the most recent transaction log dump, Tape 6.
4. Use **online database** to set the database status to "online."

Suspending and Resuming Updates to Databases

quiesce database hold allows you to block updates to one or more databases while you perform a disk unmirroring or external copy of each database device.

Because no writes are performed during this time, the external copy (the secondary image) of the database is identical to the primary image. While the database is in the quiescent state,

read-only queries to operations on the database are allowed. To resume updates to the database, issue **quiesce database release** when the external copy operation has completed. You can load the external copy of the database onto a secondary server, ensuring that you have a transactionally consistent copy of your primary image. You can issue **quiesce database hold** from one **isql** connection and then log in with another **isql** connection and issue **quiesce database release**. See the *Reference Manual: Commands*.

Note: *tag_name* must conform to the rules for identifiers. You must use the same *tag_name* for both **quiesce database...hold** and **quiesce database...release**.

For example, to suspend updates to the `pubs2` database, enter:

```
quiesce database pubs_tag hold pubs2
```

SAP ASE writes messages similar to the following to the error log:

```
QUIESCE DATABASE command with tag pubs_tag is being executed by
process 9.
Process 9 successfully executed QUIESCE DATABASE with HOLD option for
tag pubs_tag. Processes trying to issue IO operation on the quiesced
database(s) will be suspended until user executes Quiesce Database
command with RELEASE option.
```

Any updates to the `pubs2` database are delayed until the database is released, at which time the updates complete. To release the `pubs2` database, enter:

```
quiesce database pubs_tag release
```

After releasing the database, you can bring up the secondary server with the `-q` parameter if you used the **for external dump** clause. Recovery makes the databases transactionally consistent, or you can wait to bring the database online and then apply the transaction log.

Guidelines for using quiesce database

The simplest way to use **quiesce database** is to make a full copy of an entire installation, which ensures that system mappings are consistent.

These mappings are carried to the secondary installation when the system databases that contain them are physically copied as part of **quiesce database hold**'s set of databases. These mappings are fulfilled when all user databases in the source installation are copied as part of the same set. **quiesce database** allows for eight database names during a single operation. If a source installation has more than eight databases, you can issue multiple instances of **quiesce database hold** to create multiple concurrent quiescent states for multiple sets of databases.

To create a new source installation, you can use almost identical scripts to create both the primary and secondary installations. The script for the secondary installation might vary in the physical device names passed to the **disk init** command. This approach requires that updates to system devices on the primary server be reflected by identical changes to the secondary server. For example, if you perform an **alter database** command on the primary server, you must also perform the same command on the secondary server using identical parameters. This approach requires that the database devices be supported by a volume manager, which can

present to both the primary and secondary servers the same physical device names for devices that are physically distinct and separate.

Your site may develop its own procedures for making external copies of database devices. However, recommends the following:

- Include the `master` database in **quiesce database**'s list of databases.
- Any process that is prevented from writing to disk in a quiesced database may be holding a resource that prevents another process from executing. For example, if a process modifies a database page in a transaction but is prevented from flushing the log pages during the **commit**, this process is probably holding an exclusive page lock, and may block a reader trying to acquire a shared page lock on the same page during the **quiesce database** operation.

  Although this problem may occur when you quiesce system databases (`sybsystemprocs`, `sybsystemdb`, or `sybsecurity` if auditing is enabled), it is most acute when you **quiesce** the master database since the master database contains many frequently used system tables. For example, if a process modifies `syslogins` with **create login** but is prevented from committing the transaction during the quiesce of the master database, the exclusive lock acquired to modify `syslogins` block any logins because these logins must acquire a shared-page lock on `syslogins`.

  Note: Quiescing the master database, or another system database, may significantly impact server performance because doing so blocks any process that attempts to update the quiesced database.
- Name devices using identical strings on both the primary and secondary servers.
- Make the environments for the `master`, `model`, and `sybsystemprocs` system databases in the primary and secondary installations identical. In particular, `sysusages` mappings and database IDs for the copied databases must be identical on the primary and secondary servers, and database IDs for both servers must be reflected identically in `sysdatabases`.
- Keep the mapping between `syslogins.suid` and `sysusers.suid` consistent in the secondary server.
- If the primary server and the secondary server share a copy of `master`, and if the `sysdevices` entry for each copied device uses identical strings, the *physname* values in both servers must be physically distinct and separate.
- Make external copies of a database using these restrictions:
  - The copy process can begin only after **quiesce database hold** has completed.
  - Every device for every database in **quiesce database**'s list of databases must be copied.
  - The external copy must finish before you invoke **quiesce database release**.
- During the interval that **quiesce database** provides for the external copy operation, updates are prevented on any disk space belonging to any database in **quiesce database**'s list of databases. This space is defined in `sysusages`. However, if space on a device is shared between a database in **quiesce database**'s list of databases and a database not in the

list, updates to the shared device may occur while the external copy is made. When you are deciding where to locate databases in a system in which you plan to make external copies, you can either:

- Segregate databases so they do not share devices in an environment where you will use **quiesce database**, or
- Plan to copy all the databases on the device (this follows the recommendation above that you make a copy of the entire installation).

- Use **quiesce database** only when there is little update activity on the databases (preferably during a moment of read-only activity). When you quiesce the database during a quiet time, not only are fewer users inconvenienced, but, depending on the third-party I/O subsystem that is to perform the external copy, there may also be less time spent synchronizing devices involved in the copy operation.
- The **mount** and **unmount** commands make it easier to move or copy databases. You can move or copy a database from one SAP ASE to another without restarting the server, as well as move or copy more than one database at a time.You can also use these commands to physically move the devices and then reactivate the databases.

  When you **unmount** a database, you remove the database and its devices from an SAP ASE. **unmount** shuts down the database and drops it from the SAP ASE; devices are also deactivated and dropped. No changes are made to the database or its pages when unmounted.

Maintaining Server Roles in a Primary and Secondary Relationship

If your site consists of two SAP ASEs, one functioning as the primary server, and the other acting as a secondary server that receives external copies of the primary server's databases, you must never mix the roles of these servers.

That is, the role each server plays can change (the primary server can become the secondary server and vice versa), but these roles cannot be simultaneously fulfilled by the same server.

Starting the Secondary Server with the -q Option

The **dataserver** `-q` option identifies the secondary server.

Do not use the `-q` option to start the primary server. Under the `-q` option, user databases that were copied during **quiesce database for external dump** stay offline until:

- You dump the transaction log for a database on the primary server with **standby access** (that is, **dump tran with standby_access**) followed by **load tran** to the copy of this database on the secondary server, and then perform **online database for standby access** on this database.
- You force the database online for read and write access by issuing **online database**. However, if you do this, the database recovery writes compensation log records, and you cannot load the transaction log without either loading the database, or making a new copy of the primary devices using **quiesce database**.

System databases come online regardless of the `-q` option, and write compensation log records for any transactions that are rolled back.

"in quiesce" Database Log Record Value Updated

-q recovery for databases copied with **quiesce database for external dump** acts much like the recovery for **load database**.

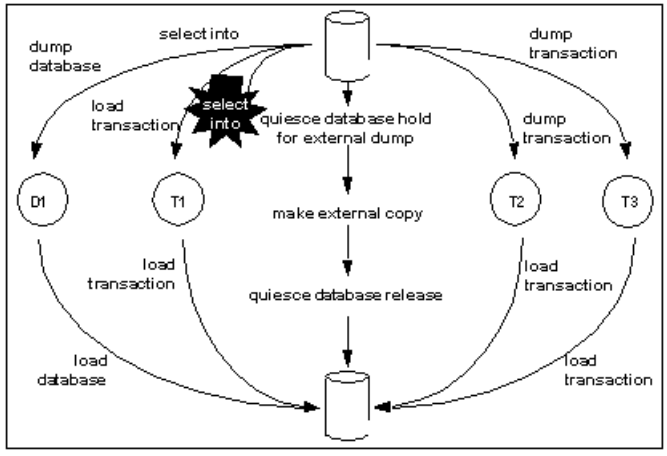
If you start the secondary server using the **-q** option of **dataserver**, SAP ASE issues a message at start-up stating that the database is `"in quiesce"` for each user database marked internally as "in quiesce." Like recovery for **load database**, **dataserver -q** internally records the address of the current last log record, so that a subsequent **load transaction** can compare this address to the address of the previous current last log record. If these two values do not match, then there has been activity in the secondary database, and SAP ASE raises error number 4306.

Updating the Dump Sequence Number

Like **dump database**, **quiesce database** updates the dump sequence numbers if there have been unlogged writes. This prevents you from using an earlier database dump or external copy as an improper foundation for a dump sequence.

For example, in the warm standby method described below, archives are produced by **dump database** (D1), **dump transaction** (T1), **quiesce database**, **dump transaction** (T2), and **dump transaction** (T3):

Figure 21: Warm Standby Dump Sequence



Typically, in an environment with logged updates and no **dump tran with truncate_only**, you could load D1, T1, T2, and T3 in turn, bypassing any **quiesce database hold**. This approach is used in a warm standby situation, where succeeding database dumps on the primary server simplify media failure recovery scenarios. On the secondary, or standby server, which is used

for decision-support systems, you may prefer continuous incremental applications of **load transaction** instead of interruptions from external copy operations.

However, if an unlogged operation occurs (for example, a **select into**, as happens above) after the **dump transaction** that produces T1, a subsequent **dump transaction to archive** is not allowed, and you must either create another dump of the database, or issue **quiesce database for external copy** and then make a new external copy of the database. Issuing either of these commands updates the dump sequence number and clears the mark that blocks the **dump transaction to archive**.

Whether or not you use the **for external dump** clause depends on how you want recovery to treat the quiescent database that would be marked as `in quiesce`.

quiesce database hold

If you issue **quiesce database** and do not use the **for external dump** clause, during the external copy operation that creates the secondary set of databases, the secondary server is not running, and recovery under `-q` does not see any copied database as "in quiesce."

It recovers each server in the normal fashion during start-up recovery; it does not recover them as **for load database** as was previously described. Subsequently, any attempt to perform a **load tran** to any of these databases is disallowed with error 4306, `"There was activity on database since last load ..."`, or with error 4305, `"Specified file '%.*s' is out of sequence ..."`

Whether or not there been unlogged activity in the primary database, the dump sequence number does not incremented by **quiesce database hold**, and the unlogged-writes bits are not cleared by **quiesce database release**.

If you attempt to run a query against a database that is quisced, SAP ASE issues error message 880:

```
Your query is blocked because it tried to write and database '%.*s'
is in quiesce state. Your query will proceed after the DBA performs
QUIESCE DATABASE RELEASE
```

The query is run once the database is no longer in a quiescent state.

quiesce database hold for external dump

When you issue **quiesce database for external dump**, the external copy of the database "remembers" that it was made during a quiescent interval, so that `-q` recovery can recover it, as happens for **load database**.

quiesce database release clears this information from the primary database. If unlogged writes have prevented **dump tran *to archive*** on the primary server, **dump tran *to archive*** is now enabled.

If unlogged writes have occurred since the previous **dump database** or **quiesce database hold for external dump** for any database in **quiesce database**'s list, the dump sequence number is updated by **quiesce database hold for external dump**, and the unlogged write

information is cleared by **quiesce database release**. The updated sequence number causes **load tran** to fail if it is applied to a target other than the external copy created under the **quiesce database** that updated it. This resembles the behavior for **dump database** of a database with unlogged writes status.

Warning! quiesce database for external dump clears the internal flag that prevents you from performing **dump transaction to** *archive_device* whether or not you actually make an external copy or perform a database dump. **quiesce database** has no way of knowing whether or not you have made an external copy. It is incumbent upon you to perform this duty. If you use **quiesce database hold for external dump** to effect a transient write protection rather than to actually perform a copy that serves as the foundation for a new dump sequence, and your application includes occasional unlogged writes, SAP ASE may allow you to create transaction log dumps that cannot be used. In this situation, **dump transaction to** *archive_device* initially succeeds, but future load transaction commands may reject these archives because they are out of sequence.

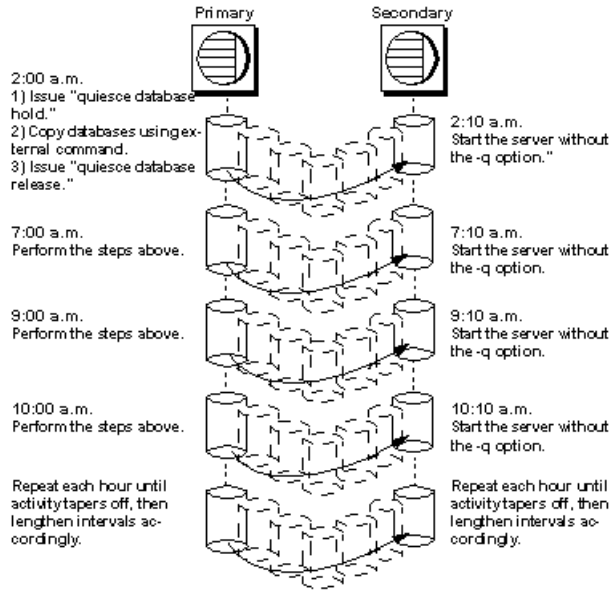
Backing up Primary Devices with quiesce database

Typically, users back up their databases with **quiesce database** either by iteratively refreshing the primary device, or via warm standby.

Both allow you to off-load decision-support applications from the online transaction processor (OLTP) server during normal operation:

- Iterative refresh of the primary device – copy the primary device to the secondary device at refresh intervals. Quiesce the database before each refresh. A system that provides weekly backups using this system is shown below:

Figure 22: Backup Schedule for Iterative Refresh Method

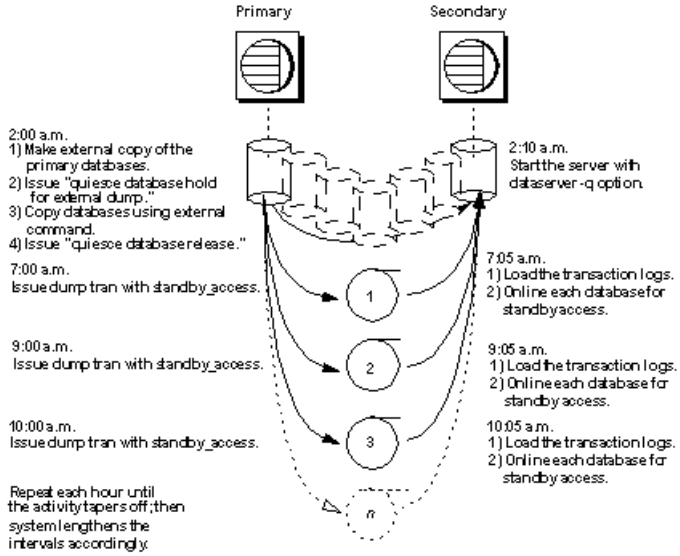


If you are using the iterative refresh method, you do not have to use the `-q` option to restart the secondary server (after a crash or system maintenance). Any incomplete transactions generate compensation log records, and the affected databases come online in the regular fashion.

- Warm standby method – allows full concurrency for the OLTP server because it does not block writes.

  After you make an external (secondary) copy of the primary database devices using the **for external dump** clause, refresh the secondary databases with periodic applications of the transaction logs with dumps from the primary server. For this method, quiesce the databases once to make an external copy of the set of databases and then refresh each periodically using a **dump tran with standby_access**. A system that uses a daily update of the primary device and then hourly backups of the transaction log is shown below.

Figure 23: Backup Schedule for Warm Standby Method



Recovery of Databases for Warm Standby Method

If you are using the warm standby method to back up a quiesced database, SAP ASE must know whether it is starting the primary or the secondary server.

Use the `-q` option of the **dataserver** command to specify that you are starting the secondary server. If you do not start the server with the `-q` option:

- The databases are recovered normally, rather than as they would be for **load database**.
- Any uncommitted transactions at the time you issue **quiesce database** are rolled back.

The recovery sequence proceeds differently, depending on whether the database is marked `in quiesce`:

- Not marked `in quiesce` – under the `-q` option, if a database is not marked `in quiesce`, it is recovered as it would be in the primary server. That is, if the database is not currently in a load sequence from previous operations, it is fully recovered and brought online. Any incomplete transactions are rolled back, and compensation log records are written during recovery.
- Marked `in quiesce` – depends if it is a user or system database:
  - User databases – user databases that are marked `in quiesce` recover in the same manner as databases recovering during **load database**. This enables **load tran** to detect any activity that has occurred in the primary database since the server was brought down. After you start the secondary server with the `-q` option, the recovery process encounters the `in quiesce` mark. SAP ASE issues a message stating that the database is in a load sequence and is being left offline. If you are using the warm

standby method, do not bring the database online for its decision-support system role until you have loaded the first transaction dump produced by a **dump tran with standby_access**. Then use **online database for standby_access**.

- System databases – system databases come fully online immediately. The `in quiesce` mark is erased and ignored.

Making Archived Copies During the Quiescent State

quiesce database hold for external dump signifies your intent to make external copies of your databases while they are quiesced.

Because these external copies are made after you issue **quiesce database hold**, the database is transactionally consistent because you are assured that no writes occurred during the interval between the **quiesce database hold** and the **quiesce database release**, and recovery can be run in the same manner as start-up recovery.

If the environment does not have unlogged updates and does not include a **dump tran with truncate_only**, you might load D1, T1, T2, and T3 in turn, bypassing any **quiesce database...hold** commands. However, if an unlogged operation (such as a **select into** shown in) occurs after the dump transaction that produces T1, dump transaction to archive is no longer allowed.

Using the **quiesce database hold for external dump** clause addresses this problem by clearing the status bits that prevent the next **dump transaction to archive** and changing the sequence number of the external copy to create a foundation for a load sequence. However, if there have been no unlogged writes, the sequence number is not incremented.

With or without the **for external** dump clause, you can make external copies of the databases. However, to apply subsequent transaction dumps from the primary to the secondary servers, include the **for external dump** clause:

```
quiesce database tag_name hold db_name [, db_name] ... [for external
dump]
```

For example:

```
quiesce database pubs_tag hold pubs2 for external dump
```

1. Issue:

   ```
   quiesce database pubs_tag hold pubs2 for external dump
   ```

2. Make an external copy of the database using the method appropriate for your site.
3. Issue:

   ```
   quiesce database pubs_tag release
   ```

Warning! Clearing the status bits and updating the sequence number enables you to perform a dump transaction whether or not you actually make an external copy after you issue **quiesce database**. SAP ASE has no way of knowing whether or not you have made an external copy during the time between **quiesce database... hold for external dump** and **quiesce database... release**. If you use the **quiesce database hold for external dump** command to effect a transient write protection rather than to actually perform a copy that can serve as the

foundation for a new dump sequence, and your application includes occasional unlogged writes, SAP ASE allows you to create transaction log dumps that cannot be used. **dump transaction to** *archive_device* succeeds, but **load transaction** rejects these archives as being out of sequence.

The mount and unmount Commands

The **mount** and **unmount** commands make it easier to move or copy databases.

You can move or copy a database from one SAP ASE to another without restarting the server (as opposed to **dump** and **load database** which copies the database to tape or disk). You can move or copy more than one database at a time using the **mount** and **unmount** commands.

You can also use these commands to physically move the devices and then reactivate the databases.

Warning! Direct mapping to a login name is not maintained within a database in SAP ASE. This means that, for every login allowed access to a database on the original SAP ASE, a corresponding login for the same `suid` must exist at the destination SAP ASE.

For permissions and protections to remain unchanged, the login maps at the secondary SAP ASE must be identical to the files on the first SAP ASE.

Using Backup Server for Backup and Recovery

Dumps and loads are performed by an Open Server program, Backup Server, running on the same machine as SAP ASE. You can perform backups over the network, using Backup Server on a remote computer and another on the local computer.

Note: Backup Server cannot dump to multidisk volumes.

Backup Server:

- Creates and loads from "striped dumps." *Dump striping* allows you to use up to 32 backup devices in parallel. This splits the database into approximately equal portions and backs up each portion to a separate device.
- Creates and loads single dumps that span several tapes.
- Dumps and loads over the network to a Backup Server running on another machine.
- Dumps several databases or transaction logs onto a single tape.
- Loads a single file from a tape that contains many database or log dumps.
- Supports platform-specific tape handling options.
- Directs volume-handling requests to the session where the **dump** or **load** command was issued or to its operator console.

- Detects the physical characteristics of the dump devices to determine protocols, block sizes, and other characteristics.

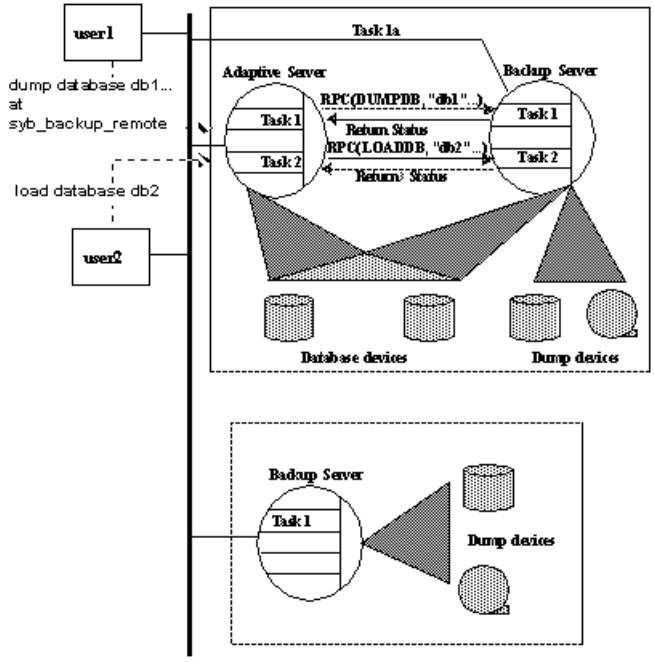
The figure below shows two users performing backup activities simultaneously on two databases:

- User1 is dumping database `db1` to a remote Backup Server.
- User2 is loading database `db2` from the local Backup Server.

Each user issues the appropriate dump or load command from an SAP ASE session. SAP ASE interprets the command and sends remote procedure calls (RPCs) to the Backup Server. The calls indicate which database pages to dump or load, which dump devices to use, and other options.

While the dumps and loads execute, SAP ASE and Backup Server use RPCs to exchange instructions and status messages. Backup Server—not SAP ASE—performs all data transfer for the **dump** and **load** commands.

Figure 24: SAP ASE and Backup Server with Remote Backup Server



When the local Backup Server receives user1's dump instructions, it reads the specified pages from the database devices and sends them to the remote Backup Server. The remote Backup Server saves the data to offline media.

Simultaneously, the local Backup Server performs user2's load command by reading data from local dump devices and writing it to the database device.

Requirements for Communicating with Backup Server

To use the **dump** and **load** commands, SAP ASE must be able to communicate with Backup Server.

These are the requirements:

- Backup Server must be running on the same machine as the SAP ASE.
- Backup Server must be listed in the `master..sysservers` table. The Backup Server entry, SYB_BACKUP, is created in `sysservers` when you install SAP ASE. Use **sp_helpserver** to see this information.
- Backup Server must be listed in the interfaces file. The entry for the local Backup Server is created when you install SAP ASE. The name of the Backup Server listed in the interfaces file must match the column `srvnetname` name for the SYB_BACKUP entry in `master..sysservers`. If you have installed a remote Backup Server on another machine, create the interfaces file on a file system that is shared by both machines, or copy the entry to your local interfaces file. The name of the remote Backup Server must be the same in both interfaces files.
- The user who starts the Backup Server process must have write permission for the dump devices. The "sybase" user, who usually starts SAP ASE and Backup Server, can read from and write to the database devices.
- SAP ASE must be configured for remote access. By default, SAP ASE is installed with remote access enabled.

See also

- *Configuring Your Server for Remote Access* on page 303

Mounting a New Volume

You may need to change tape volumes during the backup and restore process.

If the Backup Server detects a problem with the currently mounted volume, it requests a volume change by sending messages to either the client or its operator console. After mounting another volume, the operator notifies the Backup Server by executing **sp_volchanged** on SAP ASE.

On UNIX systems, the Backup Server requests a volume change when the tape capacity has been reached. The operator mounts another tape and then executes **sp_volchanged**.

Table 9. Changing tape volumes on a UNIX system

Sequence	Operator using isql	SAP ASE	Backup Server
1	Issues the **dump database** command		
2		Sends dump request to Backup Server	
3			Receives dump request message from SAP ASE Sends message for tape mounting to operator Waits for operator's reply
4	Receives volume change request from Backup Server Mounts tapes Executes **sp_volchanged**		
5			Checks tapes If tapes are okay, begins dump When tape is full, sends volume change request to operator
6	Receives volume change request from Backup Server Mounts tapes Executes **sp_volchanged**		
7			Continues dump When dump is complete, sends messages to operator and SAP ASE
8	Receives message that dump is complete Removes and labels tapes	Receives message that dump is complete Releases locks Completes the **dump database** command	

Starting and Stopping Backup Server

Starting Backup Server depends on the platform you are running.

Most UNIX systems use the **startserver** utility to start Backup Server on the same machine as SAP ASE.

On Windows, start Backup Server from SAP Control Center. See the configuration documentation for your platform for information about starting Backup Server.

Use **shutdown** to shut down a Backup Server regardless of your platform. See *System Administration Guide: Volume 1 > Diagnosing System Problems* and *System and Optional Databases* and the *Reference Manual: Commands*.

Configuring Your Server for Remote Access

By default, the **remote access** configuration parameter is set to 1 when you install SAP ASE, which allows the server to execute remote procedure calls to the Backup Server.
For security reasons, you may want to disable remote access except when dumps and loads are taking place. To disable remote access, use:

```
sp_configure "allow remote access", 0
```

Before you perform a dump or load, reenable remote access:

```
sp_configure "allow remote access", 1
```

allow remote access is dynamic and does not require a restart of SAP ASE to take effect. Only a system security officer can set **allow remote access**.

See also

- *Requirements for Communicating with Backup Server* on page 301

Choosing Backup Media

Tapes are preferred as dump devices, since they permit a library of database and transaction log dumps to be kept offline.

Large databases can span multiple tape volumes. On UNIX systems, Backup Server requires nonrewinding tape devices for all dumps and loads.

For a list of supported dump devices, see the configuration documentation for your platform.

Protecting Backup Tapes from Being Overwritten

The **tape retention in days** configuration parameter determines how many days' backup tapes are protected from being overwritten. The default value of **tape retention in days** is 0, which means that backup tapes can be overwritten immediately.
Use **sp_configure** to change the **tape retention in days** value. The new value takes effect the next time you restart SAP ASE:

```
sp_configure "tape retention in days", 14
```

Both **dump database** and **dump transaction** provide a **retaindays** option that overrides the **tape retention in days** value for that dump.

Dumping to Files or Disks

In general, SAP recommends that you do not dump a file or disk. If the disk or computer containing that file fails, there may be no way to recover the data.

On UNIX and PC systems, the entire `master` database dump must fit into a single volume. On these systems, dumping to a file or disk is your only option if the `master` database is too large to fit on a single tape volume, unless you have a second SAP ASE that can issue **sp_volchanged** requests.

You can copy dumps to a file or disk to tape for offline storage, but these tapes must be copied back to an online file before they can be read by SAP ASE. Backup Server cannot directly read a dump that is made to a disk file and then copied to tape.

Creating Logical Device Names for Local Dump Devices

If you are dumping to or loading from local devices (that is, if you are not performing backups over a network to a remote Backup Server), you can specify dump devices either by providing their physical locations or by specifying their logical device names.

In the latter case, you may want to create logical dump device names in the `sysdevices` system table of the `master` database.

Note: If you are dumping to or loading from a remote Backup Server, you must specify the absolute path name of the dump device. You cannot use a logical device name.

The `sysdevices` table stores information about each database and backup device, including its *physical_name* (the actual operating system device or file name) and its *device_name* (or logical name, known only within SAP ASE). On most platforms, SAP ASE has one or two aliases for tape devices installed in `sysdevices`. The physical names for these devices are common disk drive names for the platform; the logical names are `tapedump1` and `tapedump2`.

When you create backup scripts and threshold procedures, use logical names, rather than physical device names, and whenever possible, modify scripts and procedures that refer to actual device names each time you replace a backup device. If you use logical device names, you can simply drop the `sysdevices` entry for the failed device and create a new entry that associates the logical name with a different physical device.

Note: Make sure that the device driver options you include with the dump command are accurate. Backup Server does not verify any device driver options you include during a dump command. For example, if you include a option that forces Backup Server to rewind a tape before use, it always rewinds the tape to the beginning instead of reading the tape from the point of the dump.

To list the backup devices for your system, run:

```
select * from master..sysdevices
  where status = 16 or status = 24
```

To list both the physical and logical names for database and backup devices, use **sp_helpdevice**:

```
sp_helpdevice tapedump1
```

```
device_name physical_name
 description
 status cntrltype vdevno vpn_low      vpn_high
 ------ --------- -------------- -------- -------
tapedump1 /dev/nrmt4
tape, 625 MB, dump device
     16         3              0        0   20000
```

Adding a Backup Device

Use **sp_addumpdevice** to add a backup device.
To use an existing logical device name for a different physical device, drop the device with **sp_dropdevice** and then add it with **sp_addumpdevice**. For example:

```
sp_dropdevice tapedump2
```

```
sp_addumpdevice "tape", tapedump2, "/dev/nrmt8", 625
```

Scheduling backups of user databases

An important part of developing a backup plan is determining how often to back up your databases.

The frequency of your backups determines how much work you lose in the event of a media failure.

Dump each user database immediately after you create it, to provide a base point, and on a fixed schedule thereafter.

SAP recommends a minimum of daily backups of the transaction log and weekly backups of the database. Many installations with large and active databases make database dumps every day and transaction log dumps every half hour or hour.

Interdependent databases—databases where there are cross-database transactions, triggers, or referential integrity—should be backed up at the same time, during a period when there is no cross-database data modification activity. If one of these databases fails and requires reloading, reload them all from these simultaneous dumps.

Warning! Always dump both databases immediately after adding, changing, or removing a cross-database constraint or dropping a table that contains a cross-database constraint.

Other Times to Back Up a Database

In addition to routine dumps, dump a database each time you upgrade a user database, create a new index, perform an unlogged operation, or run the **dump transaction with no_log** or **dump transaction with truncate_only** command.

- Dumping a user database after upgrading – after you upgrade a user database to the current version of SAP ASE, dump the newly upgraded database to create a dump that is compatible with the current release. A **dump database** must occur on upgraded user databases before a **dump transaction** is permitted.
- Dumping a database after creating an index – when you add an index to a table, **create index** is recorded in the transaction log. As it fills the index pages with information, however, SAP ASE does not log the changes. If your database device fails after you create an index, **load transaction** may take as long to reconstruct the index as **create index** took to build it. To avoid lengthy delays, dump each database immediately after creating an index on one of its tables.
- Dumping a database after unlogged operations – SAP ASE writes the data for the following commands directly to disk, adding no entries (or, in the case of **bcp**, minimal entries) in the transaction log:
  - Unlogged **writetext**
  - **select into** on a permanent table
  - Fast bulk copy (**bcp**) into a table with no triggers or indexes

    You cannot recover any changes made to the database after issuing one of these commands. To ensure that these commands are recoverable, issue a **dump database** command immediately before executing any of these commands
- Dumping a database when the log has been truncated – **dump transaction with truncate_only** and **dump transaction with no_log** remove transactions from the log without making a backup copy.

  To ensure recoverability, dump the database each time you run either command because of lack of disk space. You cannot copy the transaction log until you have done so.

  If the **trunc log on chkpt** database option is set to **true**, and the transaction log contains 50 rows or more, SAP ASE truncates the log when an automatic checkpoint occurs. If this happens, dump the entire database—not the transaction log—to ensure recoverability.

See also

- *Upgrading a User Database* on page 287

Scheduling Backups of master

Current backups of the `master` database are used to recover from failures that affect that database. If you do not have a current backup of `master`, you may have to reconstruct vital system tables while you are under pressure to get your databases up and running again.

Dump the master Database After Each Change

Although you can restrict the creation of database objects in `master`, commands such as **create login** and **drop login**, and **alter login** allow users to modify system tables in the database. Back up the `master` database frequently to record these changes.

Back up the `master` database after each command that affects disks, storage, databases, or segments. Always back up `master` after issuing any of the following commands or system procedures:

- **disk init**, **sp_addumpdevice**, or **sp_dropdevice**
- Disk mirroring commands
- The segment system procedures **sp_addsegment**, **sp_dropsegment**, or **sp_extendsegment**
- **create procedure** or **drop procedure**
- **sp_logdevice**
- **sp_configure**
- **create database** or **alter database**

Save Scripts and System Tables

Save the scripts containing all of your **disk init**, **create database**, and **alter database** commands and make a hard copy of your `sysdatabases`, `sysusages`, and `sysdevices` tables each time you issue one of these commands.

You cannot use the **dataserver** command to automatically recover changes that result from these commands. If you keep your scripts—files containing Transact-SQL statements—you can run them to re-create the changes. Otherwise, you must reissue each command against the rebuilt `master` database.

Keep a hard copy of `syslogins`. When you recover `master` from a dump, compare the hard copy to your current version of the table to be sure that users retain the same user IDs.

For information on the exact queries to run against the system tables, see *System Administration Guide: Volume 1 > System and Optional Databases*.

Truncate the master Database Transaction Log

Since the `master` database transaction log is on the same database devices as the data, you cannot back up its transaction log separately.

You cannot move the log of the `master` database. You must always use **dump database** to back up the master database. Use **dump transaction** with the **truncate_only** option periodically (for instance, after each database dump) to purge the transaction log of the `master` database.

Avoid Volume Changes and Recovery

When you dump the `master` database, be sure that the entire dump fits on a single volume, unless you have more than one SAP ASE that can communicate with your Backup Server.

You must start SAP ASE in single-user mode before loading the `master` database. This does not allow a separate user connection to respond to Backup Server volume change messages during the load. Since `master` is usually small in size, placing its backup on a single tape volume is typically not a problem.

Scheduling Backups of the model Database

Keep a current database dump of the `model` database.

Each time you make a change to the `model` database, make a new backup. If `model` is damaged and you do not have a backup, you must reenter all the changes you have made to restore `model`.

Truncate the model Database's Transaction Log

`model`, like `master`, stores its transaction log on the same database devices as the data. Always use **dump database** to back up the `model` database and **dump transaction** with **truncate_only** to purge the transaction log after each database dump.

Schedule Backups of the sybsystemprocs Database

The `sybsystemprocs` database stores only system procedures. Restore this database by running the **installmaster** script, unless you make changes to the database.

If you change permissions on some system procedures, or create your own system procedures in `sybsystemprocs`, your two recovery choices are:

- Run **installmaster**, then reenter all of your changes by re-creating your procedures or by reexecuting the **grant** and **revoke** commands.
- Back up `sybsystemprocs` each time you make a change to it.

Like other system databases, `sybsystemprocs` stores its transaction log on the same device as the data. You must always use **dump database** to back up `sybsystemprocs`. By default, the **trunc log on chkpt** option is set to **true** (on) in `sybsystemprocs`, so you should not need to truncate the transaction log. If you change this database option, truncate the log when you dump the database.

If you are running on a UNIX system or PC, and you have only one SAP ASE that can communicate with your Backup Server, be sure that the entire dump of `sybsystemprocs`

fits on a single dump device. Signaling volume changes requires **sp_volchanged**, and you cannot use this procedure on a server while `sybsystemprocs` is in the process of recovery.

Configuring SAP ASE for Simultaneous Loads

SAP ASE can simultaneously perform multiple **load** and **dump** commands.

Loading a database requires one 16K buffer for each active database load. By default, SAP ASE is configured for six simultaneous loads.

To perform more loads simultaneously, a system administrator can increase the value of number of large I/O buffers:

```
sp_configure "number of large i/o buffers", 12
```

This parameter requires you to restart SAP ASE. These buffers are not used for **dump** commands or for **load transaction**. See *System Administration Guide: Volume 1 > Setting Configuration Parameters*.

Gather Backup Statistics

Use **dump database** to make several practice backups of an actual user database and **dump transaction** to back up a transaction log. Recover the database with **load database** and apply successive transaction log dumps with **load transaction**.

Keep statistics on how long each dump and load takes and how much space it requires. The more closely you approximate real-life backup conditions, the more meaningful your predictions are.

After you have developed and tested your backup procedures, commit them to paper. Determine a reasonable backup schedule and adhere to it. If you develop, document, and test your backup procedures ahead of time, you are much better prepared to get your databases online if disaster strikes.

CHAPTER 14 Backing Up and Restoring User Databases

Regular and frequent backups are your only protection against database damage that results from database-device failure.

If the Tivoli Storage Manager (TSM) is supported at your site, see also *Using Backup Server with IBM Tivoli Storage Manager*. However, most of the syntax and usage information in this chapter is relevant to sites supporting TSM.

The **dump database**, **dump transaction**, **load database**, and **load transaction** commands have parallel syntax. Routine dumps and loads require the name of a database and at least one dump device. The commands can also include these options:

- **compression=** to compress your dump files to a local file
- **at *server_name*** to specify the remote Backup Server
- **density**, **blocksize**, and **capacity** to specify tape storage characteristics
- **dumpvolume** to specify the volume name of the ANSI tape label
- **file = *file_name*** to specify the name of the file to dump to or load from
- **stripe on *stripe_device*** to specify additional dump devices
- **dismount**, **unload**, **init**, and **retaindays** to specify tape handling
- **notify** to specify whether Backup Server messages are sent to the **client** that initiated the dump or load, or to the **operator_console**

Note: When you dump a user database, its database options are not dumped because they are stored in the `sysdatabases` table of the `master` databases. This is not a problem if you load a previously dumped database onto itself, because rows in `sysdatabases` describing this database still exist in `master`. However, if you drop the database before you perform the **load database**, or if you load the database dump on a new server, these database options are not restored. To restore the image of a user database, you must also re-create the database options.

Use **dump transaction with no_log** if there is insufficient free space on the device to successfully a **dump transaction** or **dump transaction with truncate_only** command.

See the *Reference Manual: Commands*.

dump and **load database** display the percentage completed while they run. **dump database** displays the percentage completed for the database you are dumping, and **load database** displays the percentage loaded for the target database.

Note: The **dump** and **load transaction** commands do not display the percent, completed.

For example, if you dump the `sybsystemprocs` database to a file named `pubs2.dump`:

```
dump database sybsystemprocs to "pubs2.dump"
```

```
Backup Server session id is:  13.  Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume
change
request from the Backup Server.Backup Server: 4.41.1.1: Creating new
disk file
/linuxkernel_eng3/Pubs/REL1502/ASE-15_0/bin/pubs2.dump.
Backup Server: 6.28.1.1: Dumpfile name 'pubs20805209785  ' section
number 1
mounted on disk file '/linuxkernel_eng3/Pubs/REL1502/ASE-15_0/bin/
pubs2.dump'
Backup Server: 4.188.1.1: Database pubs2: 876 kilobytes (46%) DUMPED.
Backup Server: 4.188.1.1: Database pubs2: 1122 kilobytes (100%)
DUMPED.
Backup Server: 3.43.1.1: Dump phase number 1 completed.
Backup Server: 3.43.1.1: Dump phase number 2 completed.
Backup Server: 3.43.1.1: Dump phase number 3 completed.
Backup Server: 4.188.1.1: Database pubs2: 1130 kilobytes (100%)
DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database pubs2).
```

When you load `pubs2.dump` into a database:

```
load database pubs2 from "pubs2.dump"
```

```
Backup Server session id is:  17.  Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume
change
request from the Backup Server.
Backup Server: 6.28.1.1: Dumpfile name 'pubs20805209785  ' section
number 1
mounted on disk file '/linuxkernel_eng3/Pubs/REL1502/ASE-15_0/bin/
pubs2.dump'
Backup Server: 4.188.1.1: Database pubs2: 1880 kilobytes (45%)
LOADED.
Backup Server: 4.188.1.1: Database pubs2: 4102 kilobytes (100%)
LOADED.
Backup Server: 4.188.1.1: Database pubs2: 4110 kilobytes (100%)
LOADED.
Backup Server: 3.42.1.1: LOAD is complete (database pubs2).
Started estimating recovery log boundaries for database 'pubs2'.
Database 'pubs2', checkpoint=(1503, 22), first=(1503, 22),
last=(1503, 22).
Completed estimating recovery log boundaries for database 'pubs2'.
Started ANALYSIS pass for database 'pubs2'.
Completed ANALYSIS pass for database 'pubs2'.
Started REDO pass for database 'pubs2'. The total number of log
records to
process is 1.
Completed REDO pass for database 'pubs2'.
Use the ONLINE DATABASE command to bring this database online; ASE
```

```
will not
bring it online automatically.
```

For **dump database**, the percentages that appear are estimates that depend on the total size of the database being dumped. However, for **load database**, the percentages that appear are estimated according to the total size of the receiving database. For example, if you load a 500MB database dump into a 100 megabyte database, the completed percentage values are estimated according to the 100MB database, not the 50MB dump.

The remainder of this chapter provides greater detail about the information specified in dump and load commands and volume change messages.

See also

- *Database Recovery with the for load Parameter* on page 136
- *Using alter database* on page 138
- *Loading a Backup of master* on page 377

Specifying the Database and Dump Device

At a minimum, all **dump** and **load** commands must include the name of the database being dumped or loaded.

Commands that dump or load data (rather than just truncating a transaction log) must also include a dump device.

See the *Reference Manual: Commands*.

Rules for Specifying Database Names

You can specify the database name as a literal, a local variable, or a parameter to a stored procedure.

If you are loading a database from a dump:

- The database must exist. You can create a database with the **for load** option of **create database**, or load it over an existing database. Loading a database always overwrites all the information in the existing database.
- You do not need to use the same database name as the name of the database you dumped. For example, you can dump the `pubs2` database, create another database called `pubs2_archive`, and load the dump into the new database.

Warning! Do not change the name of a database that contains primary keys for references from other databases. If you must load a dump from such a database and provide a different name, first drop the references to it from other databases.

Rules for Specifying Dump Devices

You can specify the dump device as a literal, a local variable, or a parameter to a stored procedure.

When you specify a dump device:

- You cannot dump to or load from the "null device" (on UNIX, `/dev/null`; not applicable to PC platforms).
- When dumping to or loading from a local device, you can use any of these forms to specify the dump device:
  - An absolute path name
  - A relative path name
  - A logical device name from the `sysdevices` system table

  The Backup Server resolves relative path names using the SAP ASE current working directory.
- When dumping or loading over the network:
  - You must specify the absolute path name of the dump device. You cannot use a relative path name or a logical device name from the `sysdevices` system table.
  - The path name must be valid on the machine on which the Backup Server is running.
  - If the name includes any characters except letters, numbers, or the underscore (_), you must enclose it in quotes.
- If you dump a transaction log using **with standby_access**, you must load the dump using **with standby_access**.

These examples use a single tape device for dumps and loads. You need not use the same device for dumps and loads.

- On UNIX:

```
dump database pubs2 to "/dev/nrmt4"
load database pubs2 from "/dev/nrmt4"
```

- On Windows:

```
dump database pubs2 to "\\.\tape0"
load database pubs2 from "\\.\tape0"
```

You can also dump to an operating system file. The following example is for Windows:

```
dump database pubs2 to "d:\backups\backup1.dat"
load database pubs2 from "d:\backupbackup1.dat"
```

Tape Device Determination by Backup Server

When you issue a **dump database** or **dump transaction** command, Backup Server checks whether the device type of the specified dump device is known (supplied and supported internally) by SAP ASE.

If the device is not a known type, Backup Server checks the tape configuration file (default location is `$SYBASE/backup_tape.cfg)` for the device configuration.

If the configuration is found, the **dump** command proceeds.

If the configuration is not found in the tape device configuration file, the **dump** command fails with this error message:

```
Device not found in configuration file. INIT needs to be specified to
configure the device.
```

To configure the device, issue the **dump database** or **dump transaction** with the **init** parameter. Using operating system calls, Backup Server attempts to determine the device's characteristics; if successful, it stores the device characteristics in the tape configuration file.

If Backup Server cannot determine the dump device characteristics, it defaults to one dump per tape. The device cannot be used if the configuration fails to write at least one dump file.

Tape configuration by Backup Server applies only to UNIX platforms.

Tape Device Configuration File

The tape device configuration file contains tape device information that is used only by the **dump** command.

The format of the file is one tape device entry per line. Fields are separated by blanks or tabs.

The tape device configuration file is created only when Backup Server is ready to write to it. When Backup Server tries to write to this file for the first time, you see:

```
Warning, unable to open device configuration file for reading.
Operating system error. No such file or directory.
```

Ignore this message. Backup Server gives this warning and then creates the file and writes the configuration information to it.

The only user interaction that is required is if the file occurs when the user receives this error message:

```
Device does not match the current configuration. Please reconfigure
this tape device by removing the configuration file entry and issuing
a dump with the INIT qualifier.
```

This means that the tape hardware configuration changed for a device name. Delete the line entry for that device name and issue a **dump** command, as instructed.

The default path name for the configuration file is `$SYBASE/backup_tape.cfg`, which you can change using the default location with the SAP installation utilities.

Compressing a Dump

The **dump** command lets you compress databases and transaction logs using Backup Server, thereby reducing your space requirements for your archived databases.

The parameters are:

- **compression = *compression_level*** – compresses to a remote server. Causes the Backup Server to use its own native compression method. SAP recommends this compression option.
- **compress::[*compression_level*::]** – compresses to a local file. Causes the Backup Server to invoke a external filter, and is supported for backward compatibility.

compression_level is a number between 0 and 9, 100, or 101. For single-digit compression levels, 0 indicates no compression, and 9 provides the highest level of compression. Compression levels of 100 and 101 provide faster, more efficient compression, with 100 providing faster compression and 101 providing better compression.

Note: The **compress::** parameter does not support compression levels 100 and 101.

See the *Reference Manual: Commands*.

The **compression=** parameter of the **dump** command allows you to reduce your space requirements for your archived databases. With SAP ASE 12.5.2 and later, the **compression=** parameter enables you to compress your dumps to a remote machine.

If you use the older **compress::** option, you need not include the compression level when you load the database dump. However, you can issue **load with listonly=full** to determine the compression level at which the dump was made.

If you use the native **compression=** option, you need not include the **compression=** option when you load the database dump.

For example, to dump the `pubs2` database into the file "`compress_file`", enter:

```
dump database pubs2 to compress_file...compression=100
```

This shows the compression levels for the `pubs2` database. These numbers are for reference only; the numbers for your site may differ depending on operating system level and configuration.

Compression Levels	Compressed File Size
Level 1	254K
Level 9	222K

Compression Levels	Compressed File Size
Level 100	324K
Level 101	314K

Compression levels 100 and 101 are less CPU-intensive than levels 0 – 9, yet provide faster compression. However, using the 100 and 101 levels may result in larger dump files. Level 100 provides faster compression, while level 101 provides more complete compression. The actual compression result depends on the contents of your files.

SAP recommends that you choose a set of compression levels based on your performance requirements. For less CPU-intensive compression, use compression level 100 and switch to level 101 based on archive space requirement. For regular compression, use compression level 6, then increase or decrease the level based on your performance requirements.

This example dumps the `pubs2` database to the remote machine called "remotemachine" and uses a compression level of 4:

```
dump database pubs2 to "/Syb_backup/mydb.db" at remotemachine
with compression ="4"
```

This example dumps the `pubs2` database to the remote machine called "remotemachine" and uses a compression level of 100:

```
dump database pubs2 to "/Syb_backup/mydb.db" at remotemachine
with compression ="100"
```

See the *Reference Manual: Commands*.

Backup Server Dump Files and Compressed Dumps

When you perform **dump database** or **dump transaction** to a tape device using an archive file that already exists, Backup Server automatically checks the header of the existing dump archive.

Note: This issues described in this section apply to the **compress::** parameter, not to the **compression=** parameter.

If the header is unreadable, Backup Server assumes that the file is a valid non-archive file, and prompts you to change the dump archive:

```
Backup Server: 6.52.1.1: OPERATOR: Volume to be overwritten on '/opt/
new_user/DUMPS/model.dmp' has unrecognized label data.
Backup Server: 6.78.1.1: EXECUTE sp_volchanged
    @session_id = 5,
    @devname = '/opt/new_user/DUMPS/model.dmp',
    @action = { 'PROCEED' | 'RETRY' |
'ABORT' },
    @vname = <new_volume_name>
```

For this reason, if you perform **dump database** or **dump transaction** to a file without the **compress::** option into an existing compressed dump archive, Backup Server does not recognize the header information of the archive because it is compressed.

The second **dump database** reports an error, and prompts you with **sp_volchanged**:

```
dump database model to 'compress::model.cmp'
go
dump database model to 'model.cmp'
go
```

To prevent this error, include the **with init** option in your subsequent **dump database** and **dump transaction** commands:

```
dump database model to 'compress::model.cmp'
go
dump database model to 'model.cmp'
   with init
go
```

Loading Compressed Dumps

If you make a dump with the native **compression=** option, it does not require any specific syntax to load.

Note: This section describes issues pertinent to the **compress::** option.

If you use **dump ... compress::** to dump a database or transaction log, you must load this dump using the **load ... compress::** option.

The partial syntax for **load database ... compress::** and **load transaction ... compress::** is:

```
load database database_name
 from compress::stripe_device
…[stripe on compress::stripe_device]…
```

```
load transaction database_name
 from compress::stripe_device
…[stripe on compress::stripe_device]…
```

The *database_name* in the syntax is the database you archived, and **compress::** invokes the decompression of the archived database or transaction log. *archive_name* is the full path to the archived database or transaction log that you are loading. If you did not include a full path when you created your dump file, SAP ASE created a dump file in the directory in which you started SAP ASE.

If you use the **compress::** option, it must be part of the **stripe on** clause for each dump device. If you use the **compression=** option, it is used once after the device list.

Note: Do not use the *compression_level* variable for the **load** command.

See the *Reference Manual: Commands*.

See also

- *Specify Additional Dump Devices with the stripe on Clause* on page 349

Cyclic Redundancy Checks for dump database

A cyclic redundancy check has been added for accidental changes to raw data for database or transaction dumps created with compression. Use the cyclic redundancy check for checking and verifying that the compression blocks can be correctly read and decompressed.

The syntax is:

```
dump database database_name to dump_device with
compression=n,verify={crc | read_after_write}
load database database_name from dump_device with verify[only]=crc
```

Where:

- **verify=crc** – indicates that you are performing a cyclic redundancy check.
- **verify=read_after_write** – Backup Server rereads every compressed block after writing and decompresses it. If Backup Server finds an error, it prints the offset in the file in which it finds the error. **verify=read_after_write** is only applicable with the **dump database** command.

This example verifies database `new_dump` before dumping it to the `mydumpdev` device:

```
dump database new_dump to mydumpdev with
compression=x,verify=read_after_write
```

This example performs a cyclic redundancy check as it loads the `new_dump` database dump:

```
load database new_dump from mydumpdev with verify[only]=crc
```

This example performs a cyclic redundancy check and rereads every compressed block before dumping database `new_dump` to the `mydumpdev` device:

```
dump database new_dump to mydumpdev with
compression=x,verify=read_after_write,verify=crc
```

Usage:

- Dumps created without the **verify=crc** parameter use the same format as earlier versions of Backup Server.
- The **verify=crc** option is ignored if the database was not originally dumped using **verify=crc**.
- You cannot load dumps that include cyclic redundancy checks with versions of Backup Server that do not include this functionality.
- **verify={crc | read_after_write}** checks are applicable only for files created using the **with compression** parameter.
- **verify=crc** works with any file type, including raw devices, disk files, tapes, pipes, or APIs. However, **verify=read_after_write** requires a 'seek back' for rereading the block, and is applicable only with raw devices and disk files.
- If you include **verify={crc | read_after_write}** parameters that are not applicable, they are ignored and no error message is raised.

Dump History File

SAP ASE maintains the history of successful and failed backups from **dump database** and **dump transaction** commands in a dump history file.

SAP ASE reads the dump history file to restore a database, and generates the **load database** and **load transaction** sequences that are required to restore the database to a specific point in time.

Each server instance has a dump history file (located in `$SYBASE/$SYBASE_ASE`) with information about all database dumps and server configuration dumps, successful or not.

Back up dump history files with this syntax, where `file_name` is the name of your dump history file:

```
dump configuration with file = dump_hist
```

The default dump history file name is dumphist.

Each line in the dump history file represents a dump record. Dumping a database to many stripe devices results in dump records for each stripe device. Dump record fields are separated by tabs.

Dump records include information about:

- Record types
- Database IDs
- Database names
- Dump types
- Total number of stripes for dump operations
- Remote Backup Server names
- Dump current sequence numbers (timestamp for the current dump)
- Dump previous sequence numbers (timestamp for the previous dump)
- Dump creation times
- Stripe names
- Dump server names
- Error numbers
- Password-protected information (Boolean value indicating whether the backup is password protected)
- Compression levels
- Highest logical page numbers (the highest logical page number in the database that was dumped)
- Status

The dump history file is read and written by SAP ASE. The user starting the server requires appropriate read and write permissions to the dump history file.

See also

- *Concurrent dump database and dump transaction Commands* on page 339

Backups for the Dump Configuration Files

Create a backup of SAP ASE configuration files in a specified dump directory.

The **dump configuration** command allows you to back up the SAP ASE configuration file, the dump history file, and the cluster configuration file.

Dump configuration options can be defined to create a database dump. Backup Server then uses the configuration to perform a database dump. You can use:

- The **dump configuration** command to create, modify, or list dump configurations, then use **dump database** or **dump transaction** with the dump configurations to perform dumps.
- The **enforce dump configuration** configuration parameter to enable dump operations with dump configurations.
- The configuration group `dump configuration` to manage user-created dump configurations.

Dumping the history file allows you to:

- Preserve the history of **dump database** and **dump transaction** commands in a dump history file that can be used later to restore databases up to a specified point in time.
- Read the dump history file and regenerate the load sequence of SQL statements necessary to restore the database. Use:

  ```
  load database with listonly=load_sql until_time = datetime
  ```
- Use the **sp_dump_history** system procedure to purge dump history records.
- Use the **enable dump history** configuration parameter to disable default updates to the dump history file at the end of every dump operation.
- Use the `dump history filename` configuration parameter to specify the name of the dump history file.

You can also use these **dump with listonly** command options:

- Use the **create_sql** option to list the sequence of **disk init**, **sp_cacheconfig**, **create database**, and **alter database** commands that are needed to create a target database with the same layout as the source database.
- Use the **load_sql** option, which uses the dump history file to generate the list of **load database** and **load transaction** commands that are needed to repopulate the database to a specified point in time.

Performing Cumulative Dumps

The **dump database** command allows you to perform a cumulative backup, in which you make a copy of all the pages in the database that have been modified since the last full database dump.

Cumulative backups let you:

- Periodically roll forward a full backup of a database without having to back up the entire database.
- Recover changes that have been done with minimal logging (such as **select into**, when full logging is not enabled, and **parallel sort**).
- Speed recovery time – recovery time is minimized when compared to a strategy that uses transaction log dumps, as most of the changes you need are already applied.
- Reduce backup size, especially in those databases that hold big read-only tables.
- Improve backup performance, especially on database loads that have substantial updates on a subset of the database pages.
- Estimate both the dump and the load time in advance.
- Incrementally back up low-durability databases.

You can perform a cumulative dump and load in SAP ASE by specifying the new **cumulative** option in the **dump database** and **load database** commands.

You can have dump-and-load sequences that include a full database dump (using **dump database** *database_name* **full**), a cumulative dump (using **dump database** *database_name* **cumulative**), and transaction log dumps (using **dump tran[saction]** *database_name*).

You can perform a cumulative dump on any database except `master` and temporary databases. This includes low-durability databases (those created with a durability other than **full**). Until SP100, only **dump database** was supported on these databases, as earlier versions did not support **load transaction**. You can now perform a cumulative dump instead of a transaction log dump and have up-to-the-minute recovery of such databases.

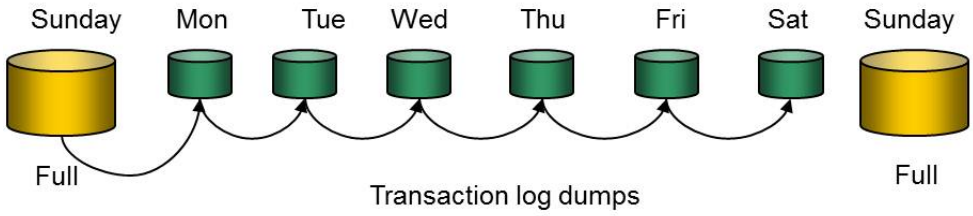
You can also perform cross-platform cumulative dumps.

Note: You cannot perform cumulative dumps in the Cluster Edition.

Dump and Load Sequences

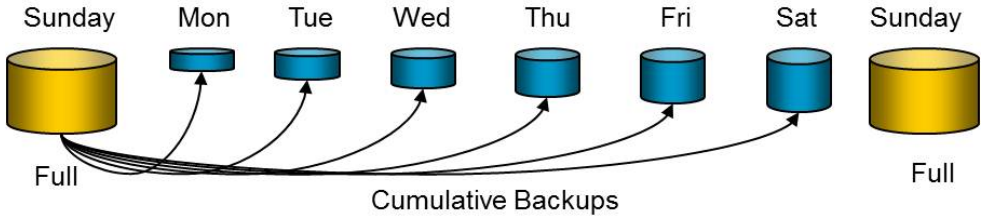
You can perform cumulative dumps instead of transaction log dumps between full database dumps.

In versions earlier than SAP ASE version 15.7 SP100, a dump sequence typically consisted of a database dump with one or more transaction logs, with the database dump and all transaction logs loaded at the same time, such as in this scenario, where a full database dump is performed once a week:



- Sunday – full database dump
- Monday – transaction log dump
- Tuesday – transaction log dump
- Wednesday – transaction log dump
- Thursday – transaction log dump
- Friday – transaction log dump
- Saturday – transaction log dump
- Sunday – full database dump

Using the cumulative dump feature, you can now perform transaction log dumps with cumulative dumps, such as:



For example, you can perform a full database dump once a week on Sundays, with cumulative dumps during the week:

- Sunday – full database dump
- Monday – cumulative dump
- Tuesday – cumulative dump
- Wednesday – cumulative dump
- Thursday – cumulative dump
- Friday – cumulative dump
- Saturday – cumulative dump
- Sunday – full database dump

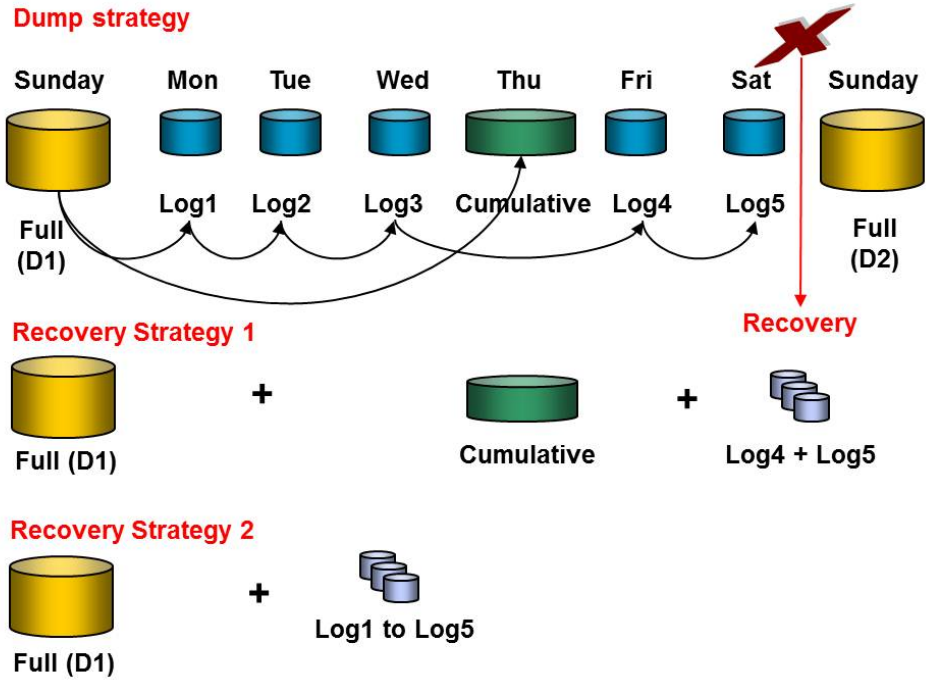
When you perform cumulative dumps, you need only load the most recent cumulative dump on top of its corresponding database dump.

However, you may not always want to perform only cumulative dumps between full database dumps. Consider these issues:

- The dump size grows, because every cumulative dump taken includes the previous one.
- If any database data devices fail, you cannot recover up to the latest point in time using transaction logs.
- If the log contains large operations such as index creation, the size of cumulative dumps can be significantly larger than the equivalent transaction log.

Note: It is not always necessary to perform a dump transaction rather than a cumulative database dump. This is because while a transaction log dump will likely be smaller, it may take significantly longer to load than if you were to load a cumulative dump after the same operation.

You can use a mixed approach that combines transaction log dumps with cumulative dumps, such as:



- Sunday – full database dump to `D1`
- Monday – transaction log dump to `Log1`
- Tuesday – transaction log dump to `Log2`
- Wednesday – transaction log dump to `Log3`
- Thursday – cumulative dump
- Friday – transaction log dump to `Log4`
- Saturday – transaction log dump to `Log5`

- Sunday – full database dump to `D2`

If you need to perform a recovery after you obtain `Log5` on Saturday, but before you can perform a new full database dump on the following Sunday, using this mixed method allows you to use one of these strategies:

- The first Sunday's full database dump to `D1`, with Thursday's cumulative dump and Friday's `Log4` and Saturday's `Log5`, or,
- The first Sunday's full database dump to `D1`, with the five transaction logs (and not using Thursday's cumulative dump).

Not all load sequences are allowed when you combine transaction log and cumulative dumps in a strategy. The following table shows the allowable load sequences for this example, where you can load the dumps listed on the left side of the table after the ones listed across the top:

```
dump database D1
dump tran T1
dump tran T2
dump cumulative C1
dump tran T3
dump tran T4
dump cumulative C2
dump database D2
dump tran T5
dump cumulative C3
```

Table 10. Allowable Load Sequences

	D1	T1	T2	C1	T3	T4	C2	D2	T5	C3
D1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
T1	Y									
T2		Y								
C1	Y	Y	Y							
T3			Y	Y						
T4					Y					
C2	Y	Y	Y	Y	Y	Y				
D2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
T5						Y	Y	Y		
C3								Y	Y	

This table does not show the minimal load sequence required. For example, the quickest way to get to a point in time that is the same as the second cumulative dump (`C2`) is to load `D1` and `C2`, rather than `D1`, `T1`, `T2`, `T3`, `T4`, and `C2`.

Partially Logged Operations and Cumulative Dumps

You can use cumulative dumps to incrementally dump the database after a partially logged operation such as **select into** (when full logging is not enabled), or parallel **create index**.

The ability to combine cumulative dumps with database dumps and transaction log dumps allows you to perform this example:

```
dump database D1
dump tran T1
dump tran T2
dump cumulative C1
dump tran T3
dump tran T4
 -- execute a partially logged operation
dump cumulative C2
dump tran T5
dump database D2
dump tran T6
dump cumulative C3
```

This table shows the allowable load sequence for the example, where you can load the dumps, where you can load the dumps listed on the left side of the table after the ones listed across the top:

Table 11. Allowable Load Sequences for Partially Logged Operations

	D1	T1	T2	C1	T3	T4	C2	T5	D2	T6	C3
D1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
T1	Y										
T2		Y									
C1	Y	Y	Y								
T3			Y	Y							
T4					Y						
C2	Y	Y	Y	Y	Y	Y					
T5							Y				
D2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
T6								Y	Y		
C3									Y	Y	

Although you cannot load cumulative dumps into a version earlier than 15.7 SP100, you may load transaction log dumps taken as part of a dump sequence that includes cumulative dumps into an earlier version if you did not perform a partially logged operation.

Warning! If you performed a partially logged operation, you cannot use the load sequence at all in an earlier version.

Restrictions

There are several restrictions that apply to cumulative dumps.

Although you cannot perform cumulative dumps in a shared-disk cluster system, you can load a cumulative dump that has been taken in a noncluster environment and load it into a database in a shared-disk cluster.

You can have dump and load sequences that include a full database dump, cumulative dumps, and transaction log dumps.

You can perform cumulative dumps on:

- In-memory databases (IMDBs) and reduced-durability databases (RDDBs) – since **load transaction** requires full logging and ordered logging, neither of which **dump transaction** performs on IMDBs and RDDBs, using cumulative dumps lets you avoid these limitations altogether.
- Databases with mixed log and data segments.

You cannot perform cumulative dumps on:

- The `master` database.
- Proxy databases.
- Temporary databases.
- (SAP ASE Cluster Edition) Shared-disk cluster environments – however, you may load a cumulative dump from in a noncluster environment into a shared-disk cluster.

You cannot load cumulative dumps into an archive database, and doing so generates an error message.

You cannot load cumulative dumps into a version of SAP ASE earlier than 15.7 SP100.

Specifying a Remote Backup Server

Use the **at** *backup_server_name* clause to send dump and load requests over the network to a Backup Server that is running on another machine.

See the *Reference Manual: Commands*

Note: The **compress::** option works only with local archives; you cannot use the *backup_server_name* option.

Sending dump and load requests over the network is ideal for installations that use a single machine with multiple tape devices for all backups and loads. Operators can be stationed at these machines, ready to service all tape change requests.

These examples dump to and load from the remote Backup Server called REMOTE_BKP_SERVER:

```
dump database pubs2 to "/dev/nrmt0" at REMOTE_BKP_SERVER
```

```
load database pubs2 from "/dev/nrmt0" at REMOTE_BKP_SERVER
```

The *backup_server_name* must appear in the interfaces file on the machine where SAP ASE is running, but does not need to appear in the `sysservers` table. The *backup_server_name* must be the same in both the local and the remote interfaces file.

The remote backup server name length limit for **dump** and **load** commands is 255 characters.

Remote Dump Host Control

Use the remote access control file to prevent remote dumps and loads, and execution of remote procedure calls (RPCs) from any client or server that is running on unauthorized servers.

Authorization to dump to, or load from, Backup Server is achieved by including the authorized hosts in the `hosts.allow` file. The default name of the file is `hosts.allow`, which is by default located in `$SYBASE`. You can change the location and file name using:

`backupserver -h` *full_path_name*`/hosts.allow`

When you start Backup Server, the location of the file is shown in the error log. For example:

```
Backup Server: 1.88.1.1: The hosts authentication file used by
the backup server is '/remote/myServer/ase157x/SMP/release/
hosts.allow'.
```

If you do not specify a file, `$SYBASE/hosts.allow` is used. If the location of the file is a relative path, the path is replaced by the absolute path using the directory in which the Backup Server has been started. For example, if you start Backup Server from `/usr/u/myServer` and Backup Server is started with:

```
backupserver -h./myhosts.allow
```

The error log shows:

```
Backup Server: 1.88.1.1: The hosts authentication file used by
the backup server is '/usr/u/myServer/./myhosts.allow'.
```

If the file `hosts.allow` does not exist, dump or load commands, or remote procedures, fail.

Note: Local dumps are not affected by this feature.

File Format

The format for `hosts.allow` is:

```
host_name_running_backupserver [ \t*][,][ \t*] host_name_trying_to_connect
```

```
host_name_running_backupserver:
hostname | hostname.domain | ipv4 address | ipv6 address
```

```
host_name_trying_to_connect:
hostname | hostname.domain | ipv4 address | ipv6 address |+

The '+' sign can be used as a wildcard to allow remote
dumps to, or loads from, any Backup Server running on
the specified host.
```

```
Example:
# Example of hosts.allow file
# Development machine imetsol1 allows access from everywhere
imetsol1 +

# Group development machine marslinuxX allow access from other
# marslinuxX machines only
marslinux1 marslinux2
marslinux1 marslinux3
marslinux2 marslinux1
marslinux2 marslinux3
marslinux3 marslinux1
marslinux3 marslinux2
```

Permissions

The recommended file permission for UNIX is no greater than `640`. For Windows, ensure that only authorized users are granted access to the file.

Error and Warning Messages

- On UNIX, if permission levels are set lower than `640`, you see a warning similar to:
  ```
  Backup Server: 1.86.1.1: Warning: The file './hosts.allow'
  has an unsafe permission mask 0664. The recommended value is
  0640.
  ```
- On Windows, if you have not established permissions, or if access is granted to everyone, you see a warning similar to:
  ```
  Backup Server: 1.87.1.1: Warning: The file './hosts.allow'
  either has no access control or one of the entries allows
  access to everyone. It is recommended that only the owner
  has permission to access the file.
  ```
- If you attempt to load to, or dump from, a remote Backup Server that does not have the appropriate access control record, you see error:
  ```
  Backup Server: 5.16.2.2: Client-Library error: Error number
  44, Layer 4, Origin 1, Severity 4: ct_connect(): protocol
  specific layer: external error: The attempt to connect to
  the server failed. Backup Server: 5.3.2.1: Cannot open a
  connection to the slave site 'REMOTE_BS'. Start the remote
  Backup Server if it is not running.
  ```
- If you attempt to execute an RPC on a remote Backup Server that does not have the appropriate access control record, you see error:
  ```
  Msg 7221, Level 14, State 2:
  ```

```
Server 's', Line 1:
Login to site 'REMOTE_BS' failed.
```

Specifying Tape Density, Block Size, and Capacity

In most cases, Backup Server uses a default tape density and block size that are optimal for your operating system; SAP recommends that you use these defaults.

You can specify a density, block size, and capacity for each dump device. You can also specify the **density**, **blocksize**, and **capacity** options in the **with** clause for all dump devices. Characteristics that are specified for an individual tape device take precedence over those that you specify using the **with** clause.

See the *Reference Manual: Commands*

Overriding the Default Density

The **dump** and **load** commands use the default tape density for your operating system. In most cases, this is the optimal density for tape dumps.

The **density** parameter has no effect on UNIX and PC platform dumps or loads.

Note: Specify tape density only when using the **init** tape handling option.

See also

- *Reinitializing a Volume Before a Dump* on page 351

Overriding the Default Block Size

The **blocksize** parameter specifies the number of bytes per I/O operation for a dump device.

By default, the dump and load commands choose the best block size for your operating system. Whenever possible, use these defaults.

You can use the **blocksize**=*number_bytes* option to override the default block size for a particular dump device. The block size must be at least one database page (2048 bytes) and must be an exact multiple of the database page size.

For UNIX systems, the block size specified on the load command is ignored. Backup Server uses the block size that was used to make the dump.

Specifying a Larger Block Size Value

If you dump to a tape using the **dump database** or **dump transaction** commands, and specify a block size value that is larger than the maximum block size of a device as determined by Backup Server, the dump or the load may fail on certain tape drives.

An operating system error message appears; for example, on an 8mm tape drive on Hewlett Packard, the error message is:

```
Backup Server: 4.141.2.22: [2] The 'write' call failed for device
'xxx' with error number 22 (Invalid argument). Refer to your
operating system documentation for further details.
```

Do not specify a block size larger than the one specified in the tape device configuration file in `$SYBASE/backup_tape.cfg`. The block size for a device is the fifth field of the line in the tape device configuration file.

This error occurs only on tape drives where tape auto config is run; that is, the device models are not hard-coded in Backup Server code.

Specifying Tape Capacity for Dump Commands

For UNIX platforms that cannot reliably detect the end-of-tape marker, you must indicate how many kilobytes can be dumped to a tape.

If you specify the physical path name of the dump device, include the **capacity** = *number_kilobytes* parameter in the dump command. If you specify the logical dump device name, the Backup Server uses the *size* parameter stored in the `sysdevices` table, unless you override it with the **capacity** = *number_kilobytes* parameter.

The specified capacity must be at least 5 database pages (each page requires 2048 bytes). SAP recommends that you specify a capacity that is slightly below the capacity rated for your device.

A general rule for calculating capacity is to use 70 percent of the manufacturer's maximum capacity for the device, and allow 30 percent for overhead (inter-record gaps, tape marks, and so on). This rule works in most cases, but may not work in all cases because of differences in overhead across vendors and devices.

Nonrewinding Tape Functionality for Backup Server

The nonrewinding tape functionality automatically positions the tape at the end of valid dump data, which saves time when you perform multiple dump operations.

Backup Server writes an end-of-file label, EOF3, at the end of every dump operation.

Tape Operations

When a new dump is performed, Backup Server performs a scan for the most recent EOF3 label. The pertinent information is saved and the tape is positioned forward to the beginning of the next file on tape. This is the new append point.

If the EOF3 label is not found or any other problem occurs, Backup Server rewinds the tape and scans forward. Any error that occurs during these steps does not abort the dump operation, but causes Backup Server to default to rewind-and-scan behavior. If the error persists during the rewind and scan, the **dump** command aborts.

Dump Version Compatibility

Backup Server activates non-rewinding logic only if the label version on the tape is greater than or equal to 5.

To activate this logic, you must execute a **dump** command with the **with init** clause. If a **dump without init** is initiated onto a volume with a label version less than 5, you are prompted to change the volume, and the dump starts on the next volume. The label version of a multivolume dump does not change in the middle of the volume set.

Table 12. Label Version Compatibility

Label version	Enabled
'3'	No
'4'	No
'5'	Yes
'6'	Yes

Apply Source Database Attributes to the Target Database

You can apply source database attributes to the target database.

You can now use **load database with listonly=create_sql** to make sure that the SQL code that is generated to create the target database into which dumps are loaded uses the same options and attributes as that of the source database that was originally dumped.

Specifically, use **load database with listonly=create_sql** to generate the SQL statements to initialize disks and create the database of an appropriate size into which the dump can be loaded.

To ensure that a database is restored exactly, both the target database and source database require the same attributes. Attributes are assigned to the source database through either **sp_dboption**, or the **create database** or **alter database** command. In versions earlier than SAP ASE 15.7 SP100, the attribute information was not stored within a database dump; this has changed as of SP100, with **load database with listonly=create_sql** storing the attributes in the database dump.

Any database attributes that had been applied to the source database using the **create database** or **alter database** commands are now included in the **create database** or **alter database** SQL that is generated with **load database with listonly=create_sql**. In addition, any options that had been applied to the source database using the **sp_dboption** stored procedure are now included in the output of **listonly=create_sql** by way of an equivalent call to **sp_dboption** to apply the same attributes to the target database.

The **create database**/**alter database** attributes included by **listonly=create_sql** are:

- **compression**
- **lob_compression**
- **inrow_lob_length**
- **dml_logging**

Note: If you use the `listonly=create_sql` option to create a database without specifying `load_sql`, SAP ASE creates the database without the `for load` option. On the other hand, if you specify `listonly=(create_sql, load_sql)`, the **create database** and **alter database** commands are generated with the `for load` clause on the assumption that the commands generated by the `load_sql` option will be used to load the database.

The **sp_dboption** attributes included by **listonly=create_sql** are:

- **abort tran on log full**
- **allow nulls by default**
- **allow wide dol rows**
- **async log service**
- **auto identity**
- **dbo use only**
- **ddl in tran**
- **deferred table allocation**
- **delayed commit**
- **enforce dump tran sequence**
- **full logging for all**
- **full logging for alter table**
- **full logging for reorg rebuild**
- **full logging for select into**
- **identity in nonunique index**
- **no chkpt on recovery**
- **no free space acctg**
- **read only**
- **scratch database**
- **select into/bulkcopy/pllsort**
- **single user**
- **trunc log on chkpt**
- **unique auto_identity index**

Generate SQL for a Different Target Database

You can generate SQL from the dump history file and dump information to create a new target database has a different name than the source database.

The dump history file records the **dump database** (both full and cumulative) and **dump transaction** commands that were executed against a database, so that the same database can be loaded to any point in time at a later stage. For example, to print the SQL used to restore a database to its most recent version, enter:

```
load database src_dbname with listonly=load_sql
```

You can use the dump history file, together with information held within a dump, to create the database into which the load sequence takes place. To print the SQL used to create the target database, enter:

```
load database src_dbname with listonly=create_sql
```

You can combine these two commands, but ensure that *src_dbname* is an exact match to the name of the database that was originally dumped:

```
load database src_dbname with listonly=(create_sql, load_sql)
```

SAP ASE uses *src_dbname* as a key to the dump history file to locate the information to generate the **create** and **load** SQL. The example output is:

```
1> load database sourcedb with listonly=(create_sql, load_sql)
2> go
disk init
     name = 'master'
   , physname = 'd:/dbs/d_master.dev'
   , size = '450M'
go
create database sourcedb
     on master = '10M'
     log on master = '10M'
with override
for load
go
load database sourcedb FROM 'd:/dbs/full.dmp'
go
load tran sourcedb FROM 'd:/dbs/tran1.dmp'
go
load tran sourcedb FROM 'd:/dbs/tran2.dmp'
go
1>
```

When the target of the load sequence is a database with a different name entirely, use:

```
load [database | transaction]
[src_dbname as target_dbname | target_dbname = src_dbname]
```

where:

- *src_dbname* – is the name of the source database that was dumped, the entries of which are in the dump history file and are used to generate the **create** and **load** SQL for the target database.
- *target_dbname* – is the name of the target database.

This example is based on the dump history created from the source database in the previous **load database** example:

```
1> load database sourcedb as targetedb
2>       with listonly=(create_sql, load_sql)
3> go
disk init
    name = 'master'
    , physname = 'd:/dbs/d_master.asecarina'
    , size = '450M'
go
create database targetdb
   on master = '10M'
   log on master = '10M'
   with override
   for load
go
load database targetdb from 'd:/dbs/full.dmp'
go
load tran targetdb from 'd:/dbs/tran1.dmp'
go
load tran targetdb from 'd:/dbs/tran2.dmp'
go
1>
```

You can specify a database mapping clause in the **load database** or **load transaction** command only if you use the **listonly=create_sql** or **listonly=load_sql** options.

This example uses the **from** clause:

```
1> load tran sourcedb as targetdb from "d:/dbs/tran1.dmp"
2>       with listonly=create_sql
3> go
disk init
    name = 'master'
    , physname = 'd:/dbs/d_master.asecarina'
    , size = '450M'
go
create database targetdb
    on master = '10M'
    log on master = '10M'
with override
for load
go
```

Specifying the Volume Name

Use the **with dumpvolume =** *volume_name* option to specify the volume name.

dump database and **dump transaction** write the volume name to the SQL tape label. **load database** and **load transaction** check the label. If the wrong volume is loaded, Backup Server generates an error message.

You can specify a volume name for each dump device. You can also specify a volume name in the **with** clause for all devices. Volume names specified for individual devices take precedence over those specified in the **with** clause.

See the *Reference Manual: Commands*

Loading from a Multifile Volume

When you load a database dump from a volume that contains multiple dump files, specify the dump file name.

If you specify only the database name, Backup Server loads the first dump file into the specified database.

For example, this command loads the first dump file from the tape into `pubs2`, regardless of whether that dump file contains data from `pubs2`:

```
load database pubs2 from "/dev/rdsk/clt3d0s6"
```

To avoid this problem, specify a unique dump file name each time you dump or load data. To get information about the dump files on a given tape, use the **listonly = full** option of **load database**.

Identifying a Dump

When you dump a database or transaction log, Backup Server creates a default file name for the dump by concatenating aspects of the database name and the date and time of the dump.

SAP ASE concatenates the:

- Last 7 characters of the database name
- 2-digit year number
- 3-digit day of the year (1 – 366)
- Number of seconds since midnight, in hexadecimal

You can override this default using the **file =** *file_name* option. The file name cannot exceed 17 characters and must conform to the file naming conventions for your operating system.

You can specify a file name for each dump device. You can also specify a file name for all devices in the **with** clause. File names specified for individual devices take precedence over those specified in the **with** clause.

When you load a database or transaction log, use the **file =** *file_name* clause to specify which dump to load from a volume that contains multiple dumps.

When loading the dump from a multifile volume, you must specify the correct file name.

The following examples use a user-defined file-naming convention. The 15-character file name, `mydb97jul141800`, identifies the database (`mydb`), the date (July 14, 1997), and the time (18:00, or 6:00 p.m.) that the dump was made. Using the **load** command advances the tape to `mydb97jul141800` before loading:

```
dump tran publications
    to "/dev/nrmt3"
load tran publications
    from "/dev/nrmt4"
    with file = "cations930590E100"
```

```
dump database mydb
    to "/dev/nrmt3"
    with file = "mydb97jul141800"
load database mydb
    from "/dev/nrmt4"
    with file = "mydb97jul141800"
```

Improving Dump or Load Performance

When you start Backup Server, use the `-m` parameter to improve the performance of the **dump** and **load** commands by configuring more shared memory for the Backup Server.

The `-m` parameter specifies the maximum amount of shared memory used by the Backup Server. You must also configure your operating system to ensure that this amount of shared memory is available to the Backup Server. When a dump or load operation completes, its shared memory segments are released.

Note: Configuring additional shared memory improves dump and load performance only if the performance limits of the hardware setup have not been reached. Increasing the value of `-m` may not result in improved performance when dumping to a slow tape device such as QIC, but it can improve performance significantly when dumping to a faster device, such as DLT.

Compatibility with Prior Versions

There are some compatibility issues between dump files and prior versions of Backup Server.

This table indicates the dump file formats that can be loaded by your currently running version and previous versions of local Backup Servers.

	New Dump File Format	Old Dump File Format
Current version of server	Yes	Yes
Earlier version of server	No	Yes

The tables below indicate the dump file formats that can be loaded by the current and prior versions of remote Backup Servers. In a remote Backup Server scenario, the master server is the Backup Server on the same machine as the database and SAP ASE, and the slave server is the Backup Server on the same remote machine as the archive device.

The first table indicates the **load** operations that work when master server is the current version of Backup Server.

	New Dump File Format	Old Dump File Format
New slave version of server	Yes	Yes
Prior slave version of server	No	Yes

This table indicates the **load** operations that work when the master server is a prior version.

	New Dump File Format	Old Dump File Format
New slave version of server	No	Yes
Prior slave version of server	No	Yes

Reducing load database Time

Use the **optimize dump for faster load** configuration parameter to optimize **dump database** for a faster **load database** time.

The time it takes to perform a **load database** (either **full** or **cumulative**), is dependent on two primary factors:

- The time it takes for Backup Server to physically copy the pages from the dump archive into the database.
- The time it takes for **load database** to recover the loaded database pages. Recovery time is largely dictated by the amount of work that needs to be redone.

By reducing the amount of work that **load database** recovery has to redo, the time it takes to perform a **load database** decreases. The amount of recovery work that needs to be redone is related to the volume of transactional work that is on-going while **dump database** (**full** or **cumulative**) copies database pages using Backup Server, from the database into the dump archive. In versions earlier than 15.7 SP121, a single iteration in which all the pages that must be copied by Backup Server to the archive is done by **dump database**.

SAP ASE supports multiple iterations. That is, at the end of the first iteration, changes that have been made to the database while the first iteration was active, are copied by Backup Server to the archive in a second iteration. The set of changes in each iteration is less than those

in the previous iteration, giving **load database** less changes to recover. A maximum of four iterations can be performed.

The **optimize dump for faster load** configuration parameter is used to configure this behavior. The syntax is:

```
sp_configure "optimize dump for faster load", <percentage>
```

When <percentage> is set to 0, a single iteration is done. This is the same behavior as in versions of SAP ASE earlier than 15.7 SP121.

When <percentage> is set to a non-zero value, it represents the minimum percentage of data pages (not log pages) in the database that have changed relative to the total number of in-use data pages in the database since the start of the current iteration in order for the next iteration to be performed (up to a maximum of 4 iterations).

For example, when <percentage> is set to a value of 2, and if the percentage of changed data pages in the database since the start of the first iteration is greater than 2% of the total in-use data pages in the database, then at the end of the first iteration of copying pages, another iteration of copying pages is performed. In this case, the second iteration of copying pages starts, and a similar test as to whether to start a third iteration is performed at the end of the second iteration.

When optimizing a dump for faster load, it will take longer to perform a **dump database**.

Concurrent dump database and dump transaction Commands

SAP ASE allows you to run a **dump transaction** concurrently with a **dump database** command, reducing the risk of losing database updates for a longer period than that established by the dump policy.

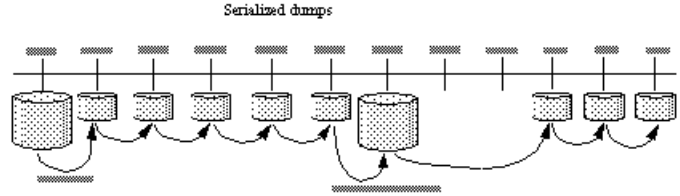
dump database works in two phases: the first phase copies the database pages to the dump archive, and the second phase copies the active part of the transaction log to the dump archive. **dump transaction** uses a single phase when it copies the active part of the transaction log to the dump archive.

dump database allows a concurrent **dump transaction** when copying the database pages (the longest phase), but does not allow a concurrent **dump transaction** when it copies the active part of the transaction log. While copying the active part of the transaction log, **dump transaction** waits for the **dump database** to finish, or vice versa, before it starts.

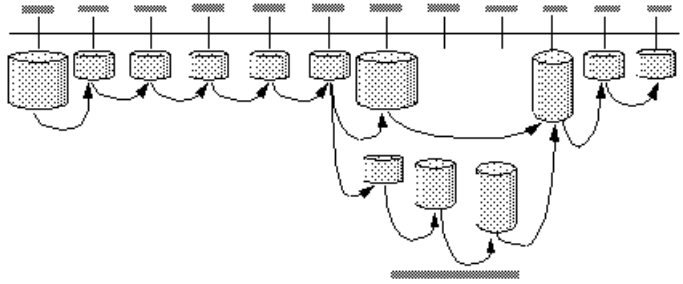
Any **dump transaction** that occurs concurrently with **dump database** (that is, completes before **dump database** starts copying the active log), cannot be loaded on top of that database dump: the dump of the transaction log belongs to the preceding load sequence.

It may be difficult to determine whether a transaction log you dumped with **dump tran** precedes the database dump (in which case it need not be loaded), or occurs after the database dump (in which case it should be loaded). Use the dump history file to resolve questions of precedence by including the transaction log or not, as appropriate.

In this example, which shows a typical daily backup strategy and uses serialized dumps, a crash at 23:15 that occurs while a **dump database** is running means that you can restore the database only to the point it was at 21:00, and more than two hours worth of database updates are lost:



However, because concurrent dumps allow you to continue dumping the transaction log while **dump database** is running, when the crash occurs at 23:15, you can restore the database to the condition it was in at 23:00 because of the transaction log dumps taken at 22:00 and at 23:00, and only 15 minutes of database updates are lost.



Note: dump transaction does not truncate the transaction log while it is running concurrently with **dump database**.

See also

- *Dump History File* on page 320

Configure SAP ASE to Run Concurrent Dumps

Enable SAP ASE to perform concurrent dumps with the **enable concurrent dump tran** configuration parameter.

With concurrent dumps, you cannot:

- Run **dump database** concurrently with a second **dump database**.
- Run **dump transaction** concurrently with a second **dump transaction**.
- Start a **dump database** command until the **dump transaction** finishes, if you start a **dump transaction** while no concurrent database dump is running.

See *System Administration Guide, Volume 1 > Setting Configuration Parameters*.

Labels Stored in Integer Format

Backup Server 12.0 and later stores the stripe number in integer format.

Versions of Backup Server earlier than 12.0 stored the 4-byte stripe number in the HDR1 label in ASCII format. These earlier versions of Backup Server cannot load a dump file that uses the newer dump format. However, Backup Server version 12.0 and later can read and write earlier versions of the dump format.

When performing a dump or load operation involving one or more remote servers, the operation aborts with an error message, if:

- The versions of one or more of the remote Backup Servers are earlier than 12.0, and the database is dumped to or loaded from more than 32 stripes, or:
- The dump file from which one or more of the Backup Servers are reading during a load is from an earlier version's format, and the number of stripes from which the database is loaded is greater than 32.

Configure Local and Remote Backup Servers

Before you perform dumps and loads, you must configure the local and remote Backup Servers at start-up by providing appropriate values for the system resources that are controlled by the command line options.

See the *Utility Guide* for a complete list of the command line options.

If your system resources are not configured properly, the dump or load may fail. For example, a remote dump to more than 25 stripes with the local and remote Backup Servers started with default configuration fails, because the maximum number of network connections that Backup Server can originate (specified by the `-N` option) is 25; however, by default, the maximum number of server connections into the remote Backup Server (specified by the `-C` option) is 30.

To configure the system to use the higher stripe limitations, set the following operating system parameters:

- Number of shared memory segments to which a process can attach
- Number of shared memory identifiers
- Swap space

If these parameters are not configured properly, when a dump is started to (or a load is started from) a large number of stripes, the operation may abort because of lack of system resources. You receive a message that Backup Server cannot create or attach to a shared memory segment and therefore, the **SYBMULTBUF** processes are terminated.

Setting Shared Memory Usage

Use the **-m** parameter to set the shared memory for Backup Server.

The syntax for starting Backup Server with the -m parameter is:

```
backupserver [-m nnn]
```

where *nnn* is the maximum amount of shared memory in megabytes that the Backup Server can use for all of its dump or load sessions.

The -m parameter sets the upper limit for shared memory usage. However, Backup Server may use less memory than specified if it detects that adding more memory will not improve performance.

Backup Server determines the amount of shared memory available for each stripe by dividing the -m value by the configured number of service threads (-P parameter).

The default value for -m is the number of service threads multiplied by 1MB. The default value for -P is 48, so the default maximum shared memory utilization is 48MB. However, Backup Server reaches this usage only if all the 48 service threads are active concurrently. The maximum value for -P is the maximum number of service threads, 12,288. See the *Utility Guide*.

The amount of shared memory per stripe available for Backup Server is proportional to the number of service threads you allocate. If you increase the maximum number of service threads, you must increase the -m value also, to maintain the same amount of shared memory per stripe. If you increase the -P value but do not increase the -m value, the shared memory allocated per stripe can decrease to the point that the dump or load cannot be processed.

Use this formula to determine how much to increase the -m value:

```
(-m value in MB) * 1024/(-P value)
```

If the value obtained by this formula is less than 128KB, Backup Server cannot start.

The minimum value for -m is 6MB. The maximum value for -m depends on operating system limits on shared memory.

If you create a dump using a Backup Server with a high shared memory value, and attempt to load the dump using a Backup Server with a lower shared memory value, Backup Server uses only the available memory, resulting in degradation of load performance.

If the amount of shared memory available per stripe at load time is less than twice the block size used at dump time, Backup Server aborts the load with an error message.

Configuring Shared Memory Dumps

Use the **memory dump compression level** configuration parameter to configure SAP ASE to compress shared memory dump files, which significantly reduces the size of them.

Enabling **memory dump compression level** can significantly reduce the size of the shared memory dump files generated by SAP ASE.

Use **sp_shmdumpconfig** to configure the shared memory dumps.

The syntax is:

```
sp_shmdumpconfig "action", type, value, max_dumps, dump_dir,
dump_file, option1, option2, option3, option4, option5
```

The *action* parameter determines how SAP ASE processes the dump. See the *Reference Manual: Commands*.

Note: Shared memory dump files are created to assist SAP Customer Support in analyzing problems in SAP ASE. Use **sp_shmdumpconfig** only under the direction of SAP Customer Support.

This example issues **sp_shmdumpconfig** with no parameters to display the current configuration for shared memory dumps:

```
sp_shmdumpconfig
```

```
Configured Shared Memory Dump Conditions
----------------------------------------

   Defaults   ---
     Maximum Dumps:          1
     Halt Engines:           Halt
     Cluster:                Local
     Page Cache:             Omit
     Procedure Cache:        Include
     Unused Space:           Omit
     Dump Directory:         $SYBASE
     Dump File Name:         Generated File Name
     Estimated File Size:    100 MB

Current number of conditions: 0
Maximum number of conditions: 10

Configurable Shared Memory Dump Configuration Settings
------------------------------------------------------
Dump on conditions: 1
Number of dump threads: 1
Include errorlog in dump file: 1
Merge parallel files after dump: 1

Server Memory Allocation
Procedure Cache  Data Caches  Server  Memory  Total Memory
---------------  -----------  -------------  ------------
          16 MB         9 MB           85 MB        108 MB
```

This example configures SAP ASE to perform a shared memory dump whenever it encounters a signal 11 (that is, a segmentation fault):

```
sp_shmdumpconfig "add", signal, 11,1,"dump_dir"
```

Setting the Maximum Number of Stripes

The maximum number of stripes that Backup Server can use is limited by the maximum number of Open Server threads it can create. Open Server imposes a maximum limit of 12K on the number of threads an application can create.

Backup Server creates one service thread for each stripe. Therefore, the maximum number of local stripes Backup Server can dump to or load from is 12,286.

As an additional limitation, Backup Server uses two file descriptors for each stripe, apart from the file descriptors associated with the error log file, interfaces file, and other system files. However, there is a per-thread limitation imposed by the operating system on the number of file descriptors. Open Server has a limitation of 1280 on the number of file descriptors that an application can keep track of.

The formula for determining the approximate maximum number of local stripes to which Backup Server can dump is:

$$\frac{(\text{The smaller of either the OS limitation or the Open Server limitation}) - 2}{2}$$

The formula for determining the approximate maximum number of remote stripes to which Backup Server can dump is:

$$\frac{(\text{The smaller of either the OS limitation or the Open Server limitation}) - 2}{3}$$

For details about the default and maximum file descriptor limits, see your operating system documentation.

Setting the Maximum Number of Network Connections

The maximum number of network connections a local Backup Server can originate is limited by Open Server to 9118.

Because of this, the maximum number of remote stripes that Backup Server can use in a single dump or load operation is 9118.

A remote Backup Server accepts a maximum of 4096 server connections at any one time. Therefore, the maximum number of remote stripes to a single remote Backup Server is 4096.

Setting the Maximum Number of Service Threads

The `-P` parameter for Backup Server configures the number of service threads Open Server creates.

The maximum number of service threads is 12,228. The minimum value is 6. The maximum number of threads equals the maximum number of stripes available. If you have started Backup Server without setting a high enough `-P` value, and you attempt to dump or load a database to a number of stripes that exceeds the number of threads, the dump or load operation fails.

Automatic Physical Database Rearrangement on Load

When loading a dump of a database that had a segregated log and data segment, SAP ASE rearranges the physical layout of the target database to ensure physical separation of the log and data segments.

How load Handles Fragments in SAP ASE 15.7 SP100 and Later

To avoid data and log fragments on the same device, SAP ASE versions 15.7 SP100 and later rearrange the physical layout of the database to accommodate for the log and data distribution in the database dump:

- SAP ASE checks that there is as much disk space for data and log on the target database as there was in the source database for data and log respectively. In versions earlier than 15.7 SP100, SAP ASE only checked that the total size of the target database was large enough to accommodate the database dump of the source. SAP ASE now the checks that there is enough disk space for log and data segments separately. If any of these segments in the target database are too small, the following error will be raised:
  ```
  The %s segment in the target database '%.*s' is too small
  (%d MB) to accommodate the %s segment from the dumped
  database (%d MB). The target database will be rearranged not
  to mix up data and log
  ```
- The target database is rearranged to prevent data and log fragments on the same device.

Example of physical database rearrangement on loading

Using the same **create database** command used earlier, the following example shows the results when there is insufficient space to accommodate the data segment when loading a database:

```
load database target_db from '/dumps/source_db.dmp'
go
Msg 3185, Level 16, State 2:
Line 1:
The data segment in the target database 'target_db' is too small (10
MB) to
accommodate the data segment from the dumped database (14 MB).
```

Because the data segment in the source database is 14 MB and only 10 MB in the target database, the space must be increased before loading the database:

```
1> alter database target_db on dev1=4
2> go
Extending database by 1024 pages (4.0 megabytes) on disk dev1
```

Now load the database:

```
1> load database target_db from '/dumps/source_db.dmp'
2> go
1> select dbid, segmap, lstart, size, vdevno, vstart, location
2> from master..sysusages where dbid=b_id("target_db")
3> go
```

The `sysusages` rows for the target database shows how SAP ASE now handles loading a database during which physical rearrangement has occurred. SAP ASE has placed the data segment (with a `segmap` of 3) on the device with `vdevno` = 1, and the log segment (with a `segmap` of 4), on the device with a `vdevno` = 2. This means that the log and data segments are now on separate devices.

```
dbid  segmap  lstart  size  vdevno  vstart  location
---- -------- ------ ------ ------- ------ ---------
 7       3        0   1536     1       3072        0
 7       4     1536    768     2       1536        0
 7       3     2304   1024     1       6144        0
 7       3     3328   1024     1       8192        0
 7       4     4352   1792     2       3072        0
```

The rearrangement of these pages and segments has left the same logical ordering, but a different physical layout. All pages in `vdevno = 1` are data pages and all pages in `vdevno = 2` are now log pages. The new pages from 4352 to 6143 that did not exist in `source_db`, because they have been created in `vdevno = 2` which is the device holding the log, have become log pages as well.

How load Handled Fragments in Releases Earlier than 15.7 SP100

A disk fragment, or fragment, is a contiguous number of database pages that are stored on a single database disk and is represented by a single row in the `sysusages` catalog. In previous releases, the way load handled disk fragments of the target database could result in undesirable results in the physical layout of the database such as:

- Fragments of the data segment and log segment being mapped to the same device
- An increased number of fragments, when compared with the fragments that made up the source database
- A database whose layout is not possible to re-create with **create database**

In versions earlier than 15.7 SP100, to move a database (the source database) to new devices on your system or to rebuild a database on the old devices, you needed to first create the new database (the target database), then load a dump of the source database onto the target database. SAP ASE would preserve the logical structure of the source database when it was loaded onto the target database. That is, every database page in a fragment of the source database occupies the same logical space in the target database, irrespective of the physical

location of the fragment in the target database to which it is copied. This can result in parts of the data segment and the log segment being physically located on the same device. In order to prevent this, you needed to create a target database that was identical to the source database by:

- Creating the target database fragments of exactly the same size as the source database and
- Creating the target database with the fragments in the same order as the source database and
- Creating the target database with fragments that belong to the same data segment or log segment as the source database

Example

The following example demonstrates how loading a dump onto a database created with fragments of different sizes, or fragments in a different order than in the source database, increases the number of fragments, and places data and log on the same device. This example is based on an SAP ASE installation using a 4K bytes page size

- Creating `source_db`

  The source database, `source_db`, was created as follows:

  ```
  1> create database source_db
  2> on dev1 = 6
  3> log on dev2 = 3
  4> go
  ```

  Later, two more data fragments on different devices were added to the database:

  ```
  1> alter database source_db
  2> on dev3 = 4,
  3> dev4 = 4
  4> go
  ```

  A select from `sysusages` shows four fragments for `source_db`:

  ```
  select dbid, segmap, lstart, size, vdevno, vstart, location
  from master..sysusages where dbid=db_id("source_db")
  go
  ```

  ```
  dbid  segmap  lstart  size  vdevno  vstart  location
  ---- -------- ------ ------ ------- ------ ---------
   4       3        0     1536    1       0       0
   4       4     1536      768    2       0       0
   4       3     2304     1024    3       0       0
   4       3     3328     1024    4       0       0
  (4 rows affected)
  ```

  Note: The device identification number is indicated by the values of `vdevno`. `dev1` has a `vdevno` value of 1, `dev2` a value of 2, `dev3` a value of 3, and `dev4` a value of 4.

- Creating `target_db`

`target_db` is created differently than `source_db`. The database is created in a single command on all of the devices at once instead of being altered onto the devices later and with a different size:

```
1> create database target_db
2> on dev1 = 10
3> log on dev2=10
6> go
```

```
1> select dbid, segmap, lstart, size, vdevno, vstart, location
2> from master..sysusages where dbid=db_id("target_db")
3> go
```

```
dbid  segmap  lstart  size  vdevno  vstart  location
---- -------- ------ ------ ------- ------ ---------
 7       3        0    2560     1      3072        0
 7       4     2560    2560     2      1536        0
```

- Loading the dump from `source_db` onto `target_db`

  The database `source_db` is dumped and loaded onto `target_db`:

  ```
  1> load database target_db from "/dumps/source_db.dmp"
  2> go
  ```

  SAP ASE keeps the logical distribution of pages from the dump when they are copied to the target, which results in the following:

  ```
  1> select dbid, segmap, lstart, size, vdevno, vstart, location
  2> from master..sysusages where dbid=b_id("target_db")
  3> go
  ```

  ```
  dbid  segmap  lstart  size  vdevno  vstart  location
  ---- -------- ------ ------ ------- ------ ---------
   7       3        0    1536     1      3072        0
   7       4     1536     768     1      6144        0
   7       3     2304     256     1      7680        0
   7       3     2560    1792     2      1536        0
   7       3     4352     768     2      5120        0
  ```

Results of example

Loading the dump of `source_db` onto `target_db` caused the following undesirable results:

- The number of disk fragments increased from two to five.
- Log and data are on the same physical device (with `vdevno` = 1). The log segment (`segmap` = 4) and the data segment (`segmap` = 3) are segregated at the logical page level, but not at the device level.

```
dbid  segmap  lstart  size  vdevno  vstart  location
---- -------- ------ ------ ------- ------ ---------
 7      3        0    1536     1      3072        0
 7      4     1536     768     1      6144        0
 7      3     2304     256     1      7680        0
 7      3     2560    1792     2      1536        0
 7      3     4352     768     2      5120        0
```

You can see from the above, a log fragment is on device `vdevno` = 1. However, there are two data fragments, (the first and third row) that are also on `vdevno` = 1. Meaning that the log segment is on the same device as the data segment.

Specify Additional Dump Devices with the stripe on Clause

Striping allows you to use multiple dump devices for a single dump or load command. Use a separate **stripe on** clause to specify the name (and, if desired, the characteristics) of each device.

Each dump or load command can have multiple **stripe on** clauses.

See the *Reference Manual: Commands*

See also

- *Loading Compressed Dumps* on page 318

Dumps to, and Loads from, Multiple Devices

Backup Server divides the database into approximately equal portions, and sends each portion to a different device.

Dumps are made concurrently on all devices, reducing the time required to dump an individual database or transaction log. Because each tape stores only a portion of the database, it is less likely that a new tape will have to be mounted on a particular device.

Warning! Do not dump the `master` database to multiple tape devices. When loading the `master` database from tape or other removable media, you cannot change volumes unless you have another SAP ASE that can respond to volume change messages.

You can use multiple devices to load a database or transaction log.

Using multiple devices decreases both the time required for the load and the likelihood of having to mount multiple tapes on a particular device.

Using Fewer Devices to Load Than to Dump

You can load a database or log even if one of your dump devices becomes unavailable between the dump and load. Specify fewer stripe clauses in the load command than you did in the **dump** command.

Note: When you dump and load over the network, you must use the same number of drives for both operations.

The following examples use three devices to dump a database but only two to load it:

```
dump database pubs2 to "/dev/nrmt0"
    stripe on "/dev/nrmt1"
    stripe on "/dev/nrmt2"
```

```
load database pubs2 from "/dev/nrmt0"
    stripe on "/dev/nrmt1"
```

After the first two tapes are loaded, a message notifies the operator to load the third.

You can also dump a database to multiple operating system files. This example is for Windows:

```
dump database pubs2 to "d:\backups\backup1.dat"
    stripe on "d:\backups\backup2.dat"
    stripe on "d:\backups\backup3.dat"
load database pubs2 from "/dev/nrmt0"
stripe on "d:\backups\backup2.dat"
    stripe on "d:\backups\backup3.dat"
```

Specifying the Characteristics of Individual Devices

Use a separate **at** *server_name* clause for each stripe device attached to a remote Backup Server.

If you do not specify a remote Backup Server name, the local Backup Server looks for the dump device on the local machine. If necessary, you can also specify separate tape device characteristics (**density**, **blocksize**, **capacity**, **dumpvolume**, and **file**) for individual stripe devices.

This example, on UNIX, uses three dump devices, each attached to the remote Backup Server REMOTE_BKP_SERVER:

```
dump database pubs2
    to "/dev/nrmt0" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt1" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt2" at REMOTE_BKP_SERVER
```

Tape Handling Options

The tape handling options, which appear in the **with** clause, apply to all devices used for the dump or load.

The options include:

- **nodismount** to keep the tape available for additional dumps or loads
- **unload** to rewind and unload the tape following the dump or load
- **retaindays** to protect files from being overwritten
- **init** to reinitialize the tape rather than appending the dump files after the last end-of-tape mark

See the *Reference Manual: Commands*.

On platforms that support logical dismounts, tapes are dismounted when a dump or load completes. Use the **nodismount** option to keep the tape mounted and available for additional dumps or loads. This command has no effect on UNIX or PC systems.

By default, both **dump** and **load** commands use the **nounload** tape handling option. On UNIX systems, this prevents the tape from rewinding after the dump or load completes. This allows you to dump additional databases or logs to the same volume, or to load additional databases or logs from that volume. Use the **unload** option for the last dump on the tape to rewind and unload the tape when the command completes.

Prevent Dump Files from Being Overwritten

The **tape retention in days** configuration parameter specifies the number of days that must elapse between the creation of a tape file and when you can overwrite it with another dump.

tape retention in days applies to all dumps requested from a single SAP ASE.

Use the **retaindays**= *number_days* option to override the **tape retention in days** parameter for a single database or transaction log dump. The number of days must be a positive integer, or zero if the tape can be overwritten immediately.

Note: tape retention in days and **retaindays** are meaningful only for disk, 1/4-inch cartridge, and single-file media. On multifile media, Backup Server checks only the expiration date of the first file.

Reinitializing a Volume Before a Dump

By default, each dump is appended to the tape following the last end-of-tape mark.

Tape volumes are not reinitialized, allowing you to dump multiple databases to a single volume. You can append new dumps only to the last volume of a multivolume dump.

Use the **init** option to overwrite any existing contents of the tape. If you specify **init**, the Backup Server reinitializes the tape without checking for:

- ANSI access restrictions
- Files that have not yet expired
- Non-SAP data

The default, **noinit**, checks for all three conditions and sends a volume change prompt if any are present.

The following example initializes two devices, overwriting the existing contents with the new transaction log dumps:

```
dump transaction pubs2
    to "/dev/nrmt0"
    stripe on "/dev/nrmt1"
    with init
```

You can also use the **init** option to overwrite an existing file, if you are dumping a database to an operating system file. This is a Windows example:

```
dump transaction pubs2
    to "d:\backups\backup1.dat"
    stripe on "d:\backups\backup2.dat"
    with init
```

See the *Reference Manual: Commands* for a description of dumping multiple databases to a single volume.

See also

- *Overriding the Default Density* on page 330

Dumping and Loading Databases with Password Protection

Protect your database dump from unauthorized loads using the **password** parameter of the **dump database** command.

You must then also include this password when you load the database.

See the *Reference Manual: Commands*.

Your password must be 6 – 30 characters long.

This example uses the password "bluesky" to protect the database dump of the `pubs2` database:

```
dump database pubs2 to "/Syb_backup/mydb.db" with passwd = "bluesky"
```

This example loads the database dump using the same password:

```
load database pubs2 from "/Syb_backup/mydb.db" with passwd =
"bluesky"
```

You can use the password-protected **dump** and **load** commands only with SAP ASE version 12.5.2 and later. If you use the password parameter on a dump of a 12.5.2 version of SAP ASE, the load fails if you try to load it on an earlier version of SAP ASE.

You can load the dump only to another server that uses the same character set. For example, if you attempt to load a dump from a server that uses an ASCII character set to a server that uses a non-ASCII character set, the load fails because the value of the ASCII password is different from the non-ASCII password.

Passwords entered by users are converted to SAP ASE local character set. Because ASCII characters generally have the same value representation across character sets, if a user's password is in an ASCII character set, the passwords for **dump** and **load** are recognized across all character sets.

Overriding the Default Message Destination

Backup Server messages inform the operator when to change tape volumes and how the dump or load is progressing. The default destination for these messages, which you can change, depends on whether the operating system offers an operator terminal feature.

The **notify** option, which appears in the **with** clause, allows you to override the default message destination for a dump or load. For this option to work, the controlling terminal or login

session from which Backup Server was started must remain active for as long as Backup Server is working; otherwise, the **sp_volchanged** message is lost.

On operating systems that offer an operator terminal feature, volume change messages are always sent to an operator terminal on the machine where Backup Server is running. Use **notify = client** to route other Backup Server messages to the terminal session where the **dump** or **load** request initiated.

On systems such as UNIX, which do not offer an operator terminal feature, messages are sent to the client that initiated the dump or load request. Use **notify = operator_console** to route messages to the terminal where the remote Backup Server was started.

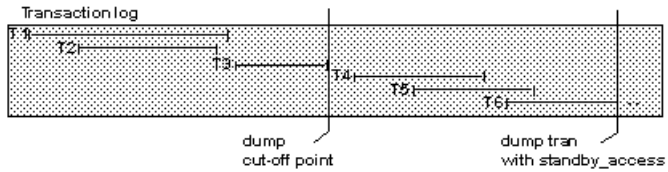
Bringing Databases Online with standby_access

Use the **with standby_access** parameter of **dump transaction** to dump only completed transactions.

It dumps the transaction log up to the point at which there are no active transactions. If you do not use **with standby_access**, the entire transaction log, including records for all open transactions is dumped.

In this example, a **dump transaction...with standby_access** command is issued at a point where transactions T1 through T5 have completed and transaction T6 is still open. The dump cannot include T5 because T6 is still open, and it cannot include T4, because T5 is still open. The dump must stop at the end of T3 (which is just before the dump cut-off point), where it includes completed transactions T1 through T3.

Figure 25: Dump Cut-Off Point for dump transaction with standby_access



Use **dump tran[saction]...with standby_access when** you are loading two or more transaction logs in sequence, and you want the database to be online between loads; for example, if you have a read-only database that gets its data by loading transaction dumps from a primary database.

1. On the primary database: **dump tran[saction]...with standby_access**
2. On the read-only database: **load tran[saction]...**
3. On the read-only database: **online database for standby_access**

Warning! If a transaction log contains open transactions, and you dump it without using **with standby_access**, SAP ASE does not allow you to load the log, bring the database online, and

then load a subsequent transaction dump. If you are going to load a series of transaction dumps, you can bring the database online only after loading a dump originally made **with standby_access**, or after loading the entire series.

The **online database** command also includes a **with standby_access** option. Use **for standby_access** to bring a database online after loading it with a dump that was made using the **with standby_access** option.

Warning! If you try to use **online database for standby_access** with a transaction log that was not dumped using the **with standby_access** option, the command fails.

See the *Reference Manual: Commands*.

Getting Information About Dump Files

se the **with headeronly** or **with listonly** option of the load commands to get information about the contents of a tape.

Note: Neither **with headeronly** nor **with listonly** loads the dump files after displaying the report.

Requesting Dump Header Information

Information about database devices is stored in the dump header.

This information, along with segment maps, are used to generate a sequence of **disk init**, **sp_cacheconfig**, and **disk init**, **create database**, and **alter database** commands for the target database during dump image creation.

Information for each database device in the dump header block includes:

- Device number
- Logical name
- Actual device size
- Physical path
- Device type
- Database device size
- Device options

In-memory database devices are configured using caches, and information about these caches is necessary to generate **disk init** and **sp_cacheconfig** commands to create the images. Cache-specific information is therefore stored in the dump header and includes:

- Cache ID
- Cache name
- Database cache size

- Actual cache size
- Device type
- Cache options

with headeronly returns the header information for a single file. If you do not specify a file name, **with headeronly** returns information about the first file on the tape.

The header indicates whether the dump is for a database or transaction log, the database ID, the file name, and the date the dump was made. For database dumps, it also shows the character set, sort order, page count, and next object ID. For transaction log dumps, the header shows the checkpoint location in the log, the location of the oldest **begin transaction** record, and the old and new sequence dates.

This example returns header information for the first file on the tape and then for the file `mydb9229510945`:

```
load database mydb
    from "/dev/nrmt4"
    with headeronly
```

```
load database mydb
    from "/dev/nrmt4"
    with headeronly, file = "mydb9229510945"
```

The output from **headeronly** looks similar to:

```
Backup Server session id is: 44. Use this value when executing the
'sp_volchanged' system stored procedure after fulfilling any volume
change request from the Backup Server.
```

```
Backup Server: 4.28.1.1: Dumpfile name 'mydb9232610BC8 ' section
number 0001 mounted on device 'backup/SQL_SERVER/mydb.db.dump'
```

```
This is a database dump of database ID 5 from Nov 21 1992 7:02PM.
```

```
Database contains 1536 pages; checkpoint RID=(Rid pageid = 0x404; row
num = 0xa); next object ID=3031; sort order ID=50, status=0; charset
ID=1.
```

Determining the Database, Device, File Name, and Date

with listonly returns a brief description of each dump file on a volume.

with listonly = full provides greater detail. Both reports are sorted by SQL tape label.

Sample output for a **load database** command **with listonly**:

```
Backup Server: 4.36.1.1: Device '/dev/nrst0':
File name: 'model9320715138  '
Create date & time: Monday, Jul 26, 2007, 23:58:48
Expiration date & time:  Monday, Jul 26, 2007, 00:00:00
Database name: 'model                          '
```

The following is sample output from **with listonly = full**

```
Backup Server: 4.37.1.1: Device '/dev/nrst0':
Label id: 'HDR1'
File name:'model9320715138  '
```

```
Stripe count:0001
Device typecount:01
Archive volume number:0001
Stripe position:0000
Generation number:0001
Generation version:00
Create date & time:Monday, Jul 26, 2007, 23:58:48
Expiration date & time:Monday, Jul 26, 2007, 00:00:00
Access code:' '
File block count:000000
Sybase id string:
'Sybase  'Reserved:'       '
Backup Server: 4.38.1.1: Device '/dev/nrst0':
Label id:'HDR2'
Record format:'F'
Max. bytes/block:55296
Record length:02048
Backup format version:01
Reserved:'   '
Database name:'model                          '
Buffer offset length:00
Reserved:'                              '
```

After listing all files on a volume, the Backup Server sends a volume change request:

```
Backup Server: 6.30.1.2: Device /dev/nrst0: Volume cataloguing
complete.
Backup Server: 6.51.1.1: OPERATOR: Mount the next volume to search.
Backup Server: 6.78.1.1: EXECUTE sp_volchanged
@session_id = 5,
          @devname = '/dev/nrst0',
          @action = { 'PROCEED' | 'RETRY' | 'ABORT' },
          @fname = '
```

The operator can use **sp_volchanged** to mount another volume and signal the volume change, or to terminate the search operation for all stripe devices.

Copying the Log After a Device Failure

Normally, **dump transaction** truncates the inactive portion of the log after copying it. Use **with no_truncate** to copy the log without truncating it.

no_truncate allows you to copy the transaction log after failure of the device that holds your data. It uses pointers in the `sysdatabases` and `sysindexes` tables to determine the physical location of the transaction log. Use **no_truncate** only if your transaction log is on a separate segment and your `master` database is accessible.

Warning! Use **no_truncate** only if media failure makes your data segment inaccessible. Do not use **no_truncate** on a database that is in use.

You can use **no_truncate** with striped dumps, tape initialization, and remote Backup Servers:

```
dump transaction mydb
to "/dev/nrmt0" at REMOTE_BKP_SERVER
with init, no_truncate,
notify = "operator_console"
```

Responding to Volume Change Requests

On UNIX and PC systems, use **sp_volchanged** to notify the Backup Server when the correct volumes have been mounted.

To use **sp_volchanged**, log in to any SAP ASE that can communicate with both the Backup Server that issued the volume change request and the SAP ASE that initiated the dump or load.

The Backup Server writes the *vname* in the ANSI tape label when overwriting an existing dump, dumping to a brand new tape, or dumping to a tape whose contents are not recognizable. During loads, the Backup Server uses the *vname* to confirm that the correct tape has been mounted. If you do not specify a *vname*, the Backup Server uses the volume name specified in the dump or load command. If neither **sp_volchanged** nor the command specifies a volume name, the Backup Server does not check this field in the ANSI tape label.

Volume Change Prompts for Dumps

Volume change prompts appear while you are dumping a database or transaction log.

Each prompt includes the possible operator actions and the appropriate **sp_volchanged** response.

- `Mount the next volume to search.`

  When appending a dump to an existing volume, the Backup Server issues this message if it cannot find the end-of-file mark.

The Operator Can	By Replying
Abort the dump	`sp_volchanged` *`session_id`*`, `*`devname`*`, abort`
Mount a new volume and proceed with the dump	`sp_volchanged` *`session_id`*`, `*`devname`*`, proceed [, `*`fname`*` [, `*`vname`*`]]`

- `Mount the next volume to write.`

  Backup Server issues this message when it reaches the end of the tape. This occurs when it detects the end-of-tape mark, dumps the number of kilobytes specified by the **capacity** parameter of the **dump** command, or dumps the *high* value specified for the device in the `sysdevices` system table.

The Operator Can	By Replying
Abort the dump	`sp_volchanged session_id, devname, abort`
Mount the next volume and proceed with the dump	`sp_volchanged session_id, devname, proceed[, fname [, vname]]`

- `Volume on device devname has restricted access (code access_code).`

  Dumps that specify the **init** option overwrite any existing contents of the tape. Backup Server issues this message if you try to dump to a tape with ANSI access restrictions without specifying the **init** option.

The Operator Can	By Replying
Abort the dump	`sp_volchanged session_id, devname, abort`
Mount another volume and retry the dump	`sp_volchanged session_id, devname, retry [, fname[, vname]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged session_id, devname, proceed[, fname[, vname]]`

- `Volume on device devname is expired and will be overwritten.`

  Dumps that specify the **init** option overwrite any existing contents of the tape. During dumps to single-file media, Backup Server issues this message if you have not specified the **init** option and the tape contains a dump whose expiration date has passed.

The Operator Can	By Replying
Abort the dump	`sp_volchanged session_id, devname, abort`
Mount another volume and retry the dump	`sp_volchanged session_id, session_id, retry[, session_id[, session_id]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged session_id, session_id, proceed[, session_id[, session_id]]`

- `Volume to be overwritten on 'devname' has not expired: creation date on this volume is creation_date, expiration date is expiration_date.`

  On single-file media, Backup Server checks the expiration date of any existing dump unless you specify the **init** option. Backup Server issues this message if the dump has not yet expired.

The Operator Can	By Replying
Abort the dump	`sp_volchanged session_id, session_id, abort`
Mount another volume and retry the dump	`sp_volchanged session_id, session_id, retry[, session_id [, session_id]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged session_id, session_id, proceed[, session_id[, session_id]]`

- `Volume to be overwritten on 'devname' has unrecognized label data.`

  Dumps that specify the **init** option overwrite any existing contents of the tape. Backup Server issues this message if you try to dump to a new tape or a tape with non-SAP data without specifying the **init** option.

The Operator Can	By Replying
Abort the dump	`sp_volchanged session_id, session_id, abort`
Mount another volume and retry the dump	`sp_volchanged session_id, session_id, retry[, session_id[, session_id]]`
Proceed with the dump, overwriting any existing contents	`sp_volchanged session_id, session_id, proceed[, session_id[, session_id]]`

Volume Change Prompts for Loads

Volume change prompts appear while you are loading a database or transaction log.

Following are the volume change prompts and possible operator actions during loads:

- `Dumpfile 'fname' section vname found instead of 'fname' section vname.`

  Backup Server issues this message if it cannot find the specified file on a single-file medium.

The Operator Can	By Replying
Abort the load	`sp_volchanged session_id, session_id, abort`

The Operator Can	By Replying
Mount another volume and try to load it	`sp_volchanged` *`session_id`*`,` *`session_id`*`, retry[,` *`session_id`*`[,` *`session_id`*`]]`
Load the file on the currently mounted volume, even though it is not the specified file (not recommended)	`sp_volchanged` *`session_id`*`,` *`session_id`*`, proceed[,` *`session_id`*`[,` *`session_id`*`]]`

- `Mount the next volume to read.`

  Backup Server issues this message when it is ready to read the next section of the dump file from a multivolume dump.

The Operator Can	By Replying
Abort the load	`sp_volchanged` *`session_id`*`,` *`session_id`*`, abort`
Mount the next volume and proceed with the load	`sp_volchanged` *`session_id`*`,` *`session_id`*`, proceed[,` *`session_id`*`[,` *`session_id`*`]]`

- `Mount the next volume to search.`

  Backup Server issues this message if it cannot find the specified file on multifile medium.

The Operator Can	By Replying
Abort the load	`sp_volchanged` *`session_id`*`,` *`session_id`*`, abort`
Mount another volume and proceed with the load	`sp_volchanged` *`session_id`*`,` *`session_id`*`, proceed[,` *`session_id`*`[,` *`session_id`*`]]`

Recovering a Database: Step-By-Step Instructions

The symptoms of media failure are as variable as the causes.

If only a single block on the disk is bad, your database may appear to function normally for some time after the corruption occurs, unless you are frequently running **dbcc** commands. If an entire disk or disk controller is bad. SAP ASE marks the database as suspect and displays a warning message. If the disk storing the `master` database fails, users cannot log in to the server, and users already logged in cannot perform any actions that access the system tables in `master`.

1. Get a current log dump of every database on the device.
2. Examine the space usage of every database on the device.

3. After you have gathered this information for all databases on the device, drop each database.
4. Use **sp_dropdevice** to drop the failed device. See the *Reference Manual: Procedures*.
5. Use **disk init** to initialize the new database devices. See *System Administration Guide: Volume 1 > Initializing Database Devices*.
6. Re-create the databases, one at a time.
7. Load the most recent database dump into each database.
8. Apply each transaction log dump in the order in which it was created.

The steps that require more detailed information than listed here are discussed in the following sections.

Getting a Current Dump of the Transaction Log

Use the **dump transaction with no_truncate** command, or the **sybdumptran** utility to get a current transaction log dump.

- Use the **sybdumptran** utility to dump the most recent transactions when the database and the server have suffered a catastrophic failure. See **sybdumptran** in the *Utility Guide*.
- Use **dump transaction with no_truncate** to get a current transaction log dump for each database on the failed device.

  For example, to get a current transaction log dump of mydb, enter:

```
dump transaction mydb
to "/dev/nrmt0" at REMOTE_BKP_SERVER
with init, no_truncate,
notify = "operator_console"
```

Examining the Space Usage

Determine which devices your database uses, how much space is allocated on each device, and whether the space is used for data, log, or both.

You can use this information when re-creating your databases to ensure that the log, data, and indexes reside on separate devices, and to preserve the scope of any user segments you have created.

Note: You can also use these steps to preserve segment mappings when moving a database dump from one server to another (on the same hardware and software platform).

If you do not use this information to re-create the device allocations for damaged databases, SAP ASE remaps the `sysusages` table after **load database** to account for discrepancies. This means that the database's system-defined and user-defined segments no longer match the appropriate device allocations. Incorrect information in `sysusages` can result in the log being stored on the same devices as the data, even if the data and the log were separate before recovery. It can also change user-defined segments in unpredictable ways, and may result in a database that cannot be created using a standard **create database** command.

1. In `master`, examine the device allocations and uses for the damaged database:

```
select segmap, size from sysusages
    where dbid = db_id("database_name")
```

2. Examine the output of the query. Each row with a `segmap` of 3 represents a data allocation; each row with a `segmap` of 4 represents a log allocation. Higher values indicate user-defined segments; treat these as data allocations, to preserve the scope of these segments. The `size` column indicates the number of blocks of data. Note the order, use, and size of each disk piece.

   For example, this is the output from a server that uses 2K logical pages translates into the sizes and uses described in the following table:

```
segmap        size
-------       --------
     3          10240
     3           5120
     4           5120
     8           1024
     4           2048
```

Table 13. Sample Device Allocation

Device Allocation	Megabytes
Data	20
Data	10
Log	10
Data (user-defined segment)	2
Log	4

Note: If the `segmap` column contains 7s, it indicates that your data and log are on the same device, and you can recover only up to the point of the most recent database dump. Do not use the **log on** option to **create database**. Just be sure that you allocate as much (or more) space than the total reported from `sysusages`.

3. Run **sp_helpdb** *database_name* for the database. This query lists the devices on which the data and logs are located:

```
name       db_size owner   dbid    created      status
-------    ------- ------  ----    -----------  -------------
mydb       46.0 MB sa        15    May 26 2005  no_options set
```

```
device_fragments  size    usage     created            free kbytes
----------------  -----   --------   ------------         -------
----
datadev1         20 MB  data only  June 7 2005 2:05PM        13850
datadev2         10 MB  data only  June 7 2005 2:05PM         8160
datadev3          2 MB  data only  June 7 2005 2:05PM         2040
logdev1           10 MB  log only   June 7 2005 2:05PM  not
applicable
```

```
logdev2             4 MB  log only   June 7 2005 2:05PM  not
applicable
```

Dropping the Databases

Use **drop database** to drop each database.

Note: If tables in other databases contain references to any tables in the database you are trying to drop, you must use **alter table** to remove the referential integrity constraints with before you can drop the database.

If the system reports errors because the database is damaged when you issue **drop database**, use:

```
dbcc dbrepair (mydb, dropdb)
```

If you are using a replicated database, use **dbcc dbrepair** to load a dump from a previous version of SAP ASE to a more current version. For example:

- Loading a dump from a production system of an earlier version of SAP ASE into a test system of the current version SAP ASE, or,
- In a warm standby application, initializing a standby database of the current version of SAP ASE with a database dump from an active database of an earlier version of SAP ASE.

See the *Error Message and Troubleshooting Guide* and *Reference Manual: Commands*.

Re-creating the Databases

Re-create each database using segment information you have collected.

Note: If you chose not to gather information about segment usage, use **create database...for load** to create a new database that is at least as large as the original.

1. Use **create database** with the **for load** option. Duplicate all device fragment mappings and sizes for each row of your database from the `sysusages` table, up to and including the first log device. Use the order of the rows as they appear in `sysusages`. (The results of **sp_helpdb** are in alphabetical order by device name, not in order of allocation.) For example, to re-create the `mydb` database allocations in the table above, enter:

   ```
   create database mydb
       on datadev1 = 20,
           datadev2 = 10
   log on logdev1 = 10
   for load
   ```

   Note: create database...for load temporarily locks users out of the newly created database, and **load database** marks the database offline for general use. This prevents users from performing logged transactions during recovery.

2. Use **alter database** with the **for load** option to re-create the remaining entries, in order. Remember to treat device allocations for user segments as you would data allocations.

In this example, to allocate more data space on `datadev3` and more log space on `logdev1`, the command is:

```
alter database mydb
    on datadev3 = "2M"
log on logdev1= "4M"
for load
```

Loading the Database

Reload the database using **load database**.

If the original database stored objects on user-defined segments (`sysusages` reports a `segmap` greater than 7) and your new device allocations match those of the dumped database, SAP ASE preserves the user segment mappings.

If you did not create the new device allocations to match those of the dumped database, SAP ASE remaps segments to the available device allocations. This remapping may also mix log and data on the same physical device.

Note: If an additional failure occurs while a database is being loaded, SAP ASE does not recover the partially loaded database, and notifies the user. You must restart the database load by repeating the **load** command.

Loading the Transaction Logs

Use **load transaction** to apply transaction log backups in the same sequence in which they were made.

SAP ASE checks the timestamps on each dumped database and transaction log. If the dumps are loaded in the wrong order, or if user transactions have modified the transaction log between loads, the load fails.

If you dumped the transaction log using **with standby_access**, you must also load the database using **standby_access**.

After you have brought a database up to date, use **dbcc** commands to check its consistency.

Loading a Transaction Log to a Point in Time

You can recover a database up to a specified point in time in its transaction log.

To do so, use the **until_time** option of **load transaction**. This is useful if, for example, a user inadvertently drops an important table; you can use **until_time** to recover the changes made to the database containing the table up to a time just before the table was dropped.

To use **until_time** effectively after data has been destroyed, you must know the exact time the error occurred. You can find this by issuing a **select getdate** at the time of the error.

1. For example, suppose a user accidentally drops an important table, and then a few minutes later you get the current time in milliseconds:

```
select convert(char(26), getdate(), 109)
--------------------------
Mar 26 2007 12:45:59:650PM
```

2. After dumping the transaction log containing the error and loading the most recent database dump, load the transaction logs that were created after the database was last dumped. Then, load the transaction log containing the error by using **until_time**; in this example, about 10 minutes earlier:

   ```
   load transaction employees_db
   from "/dev/nrmt5"
   with until_time = "Mar 26 2007 12:35:59: 650PM"
   ```

   After you load a transaction log using **until_time**, SAP ASE restarts the database's log sequence. This means that until you dump the database again, you cannot load subsequent transaction logs after the **load transaction** using **until_time**. You must dump the database before you can dump another transaction log.

Bringing the Databases Online

After you have applied all transaction log dumps to a database, use **online database** to make it available for use.

For example, to bring the `employees_db` database online, enter:

```
online database employees_db
```

Replicated Databases

You cannot bring replicated databases online until the logs are drained.

If you try to bring a replicated database online before the logs are drained, SAP ASE issues:

```
Database is replicated, but the log is not yet drained. This database
will come online automatically after the log is drained.
```

When Replication Server, via the Log Transfer Manager (LTM), drains the log, **online database** is automatically issued.

The load sequence for loading replicated databases is: **load database**, **replicate**, **load transaction**, **replicate**, and so on. At the end of the load sequence, issue **online database** to bring the databases online. Databases that are offline because they are in a load sequence are not automatically brought online by Replication Server.

Warning! Do not issue **online database** until all transaction logs are loaded.

Before you upgrade replicated databases to the current version of SAP ASE, the databases must be online. Refer to the installation documentation for your platform for upgrade instructions for SAP ASE users that have replicated databases.

Loading Database Dumps from Older Versions

During most upgrades, all databases associated with a server are automatically upgraded. Major upgrade, for example, from version 12.5.x to version 15.5, require that you run the **preupgrade** and **upgrade** utilities to complete the upgrade.

As a result, database and transaction log dumps created with an earlier version of SAP ASE must be upgraded before they can be used with the current version of SAP ASE.

SAP ASE provides an automatic upgrade mechanism—on a per-database basis—for upgrading a database or transaction log made with Backup Sever to the current SAP ASE version, thus making the dump compatible for use. This mechanism is entirely internal to SAP ASE, and requires no external programs. It provides the flexibility of upgrading individual dumps as needed.

However, these tasks are not supported by the automatic upgrade functionality:

- Loading an older version of the `master` database. That is, if you upgraded SAP ASE to the current version, you cannot load a dump of the `master` database from which you upgraded.
- Installing new or modified stored procedures. Continue to use **installmaster**.

Upgrading a Dump to the Current Version of SAP ASE

SAP ASE uses the dump header to determine from which version of SAP ASE it is loading.

After the dump header is read, and before Backup Server begins the load, the database is marked offline by **load database** or **load transaction**. This makes the database unavailable for general use (queries and **use** *database* are not permitted), provides the user greater control over load sequences, and eliminates the possibility that other users will accidentally interrupt a load sequence.

1. Use **load database** and **load transaction** to load the dump to be upgraded.
2. Use **online database**, after the dump has successfully loaded, to activate the upgrade process.

   Note: Do not issue **online database** until after all transaction dumps are loaded.

   For dumps loaded from versions 12.0 and later, **online database** activates the upgrade process to upgrade the dumps just loaded. After the upgrade is successfully completed, SAP ASE places the database online, and the database is ready for use.

   For dumps loaded from the version of SAP ASE you are currently running, no upgrade process is activated. You must still issue **online database** to place the database online—**load database** marks it as offline.

   Each upgrade step produces a message stating what it is about to do.

An upgrade failure leaves the database offline and produces a message stating that the upgrade failed and the user must correct the failure.

See the *Reference Manual: Commands*.

3. After successful execution of **online database**, use **dump database**. The database must be dumped before a **dump transaction** is permitted. A **dump transaction** on a newly created or upgraded database is not permitted until a successful **dump database** has occurred.

The Database Offline Status Bit

You can determine whether a database is offline by using **sp_helpdb**. If this bit is set, it indicates that the database is offline, and therefor, unavailable for general use.

When a database is marked offline by **load database**, a status bit in the `sysdatabases` table is set and remains set until the successful completion of **online database**.

The "database offline" status bit works in combination with any existing status bits. It augments the following status bit to provide additional control:

- In recovery

The "database offline" status bit overrides these status bits:

- DBO use only
- Read only

These status bits override the "database offline" status bit:

- Began upgrade
- Bypass recovery
- In load
- Not recovered
- Suspect
- Use not recovered

Although the database is not available for general use, you can use these commands when the database is offline:

- **dump database** and **dump transaction**
- **load database** and **load transaction**
- **alter database on device**
- **drop database**
- **online database**
- **dbcc** diagnostics (subject to **dbcc** restrictions)

Version Identifiers and Automatic Upgrade

The automatic upgrade provides version identifiers for SAP ASE, databases, and log record formats.

Including the:

- Configuration upgrade version ID – shows the current version of SAP ASE; it is stored in the `sysconfigures` table. **sp_configure** displays the current version of SAP ASE as "upgrade version."
- Upgrade version indicator – shows the current version of a database and is stored in the database and dump headers. The SAP ASE recovery mechanism uses this value to determine whether the database should be upgraded before being made available for general use.
- Log compatibility version specifier – differentiates logs from earlier and later versions of SAP ASE by showing the format of log records in a database, database dump, or transaction log dump. This constant is stored in the database and dump headers and is used by SAP ASE to detect the format of log records during recovery.

Cache Bindings and Loading Databases

If you dump a database and load it onto a server with different cache bindings, you may want to load the database onto a different server for tuning or development work, or you may need to load a database that you dropped from a server whose cache bindings have changed since you made the dump.

When you bring a database online after recovery or by using **online database** after a load, SAP ASE verifies all cache bindings for the database and database objects. If a cache does not exist, SAP ASE writes a warning to the error log, and the binding in `sysattributes` is marked as invalid. Here is an example of the message from the error log:

```
Cache binding for database '5', object '208003772', index '3' is
being marked invalid in Sysattributes.
```

Invalid cache bindings are not deleted. If you create a cache of the same name and restart SAP ASE, the binding is marked as valid and the cache is used. If you do not create a cache with the same name, you can bind the object to another cache, or allow it to use the default cache.

In the following sections, which discuss cache binding topics, destination server refers to the server where the database is being loaded, and original server refers to the server where the dump was made.

If possible, re-create caches that have the same names on the destination server as the bindings on the original server. You may want to configure pools in exactly the same manner if you are using the destination database for similar purposes or for performance testing and development that may be ported back to the original server. If you are using the destination

database for decision support or for running **dbcc** commands, you may want to configure pools to allow more space in 16K memory pools.

Databases and Cache Bindings

Binding information for databases is stored in `master..sysattributes`.

No information about database binding is stored in the database itself. If you use **load database** to load the dump over an existing database that is bound to a cache, and you do not drop the database before you issue the load command, this does not affect the binding.

If the database that you are loading was bound to a cache on the original server, you can:

- Bind the database on the destination server to a cache configured for the needs on that server, or
- Configure pools in the default data cache on the destination server for the needs of the application there, and do not bind the database to a named data cache.

Database Objects and Cache Bindings

Binding information for objects is stored in the `sysattributes` table in the database itself.

If you frequently load the database onto the destination server, the simplest solution is to configure caches of the same name on the destination server.

If the destination server is not configured with caches of the same name as the original server, bind the objects to the appropriate caches on the destination server after you bring the database online, or be sure that the default cache is configured for your needs on that server.

Checking on Cache Bindings

Use **sp_helpcache** to display the cache bindings for database objects, even if the cache bindings are invalid.
The following SQL statements reproduce cache binding commands from the information in a user database's `sysattributes` table:

```
/* create a bindcache statement for tables */

select "sp_bindcache "+ char_value + ", "
    + db_name() + ", " + object_name(object)
from sysattributes
where class = 3
    and object_type = "T"
```

```
/* create a bindcache statement for indexes */

select "sp_bindcache "+ char_value + ", "
    + db_name() + ", "   + i.name
from sysattributes, sysindexes i
where class = 3
    and object_type = "I"
    and i.indid = convert(tinyint, object_info1)
    and i.id = object
```

Cross-Database Constraints and Loading Databases

If you use the **references** constraint of **create table** or **alter database** to reference tables across databases, you may encounter problems when you try to load a dump of one of these databases.

- If tables in a database reference a dumped database, referential integrity errors result if you load the database with a different name or on a different server from where it was dumped. To change the name or location of a database when you reload it, use **alter database** in the referencing database to drop all external referential integrity restraints before you dump the database.
- Loading a dump of a referenced database that is of an earlier version than the referencing database may cause consistency issues or data corruption. As a precaution, each time you add or remove a cross-database constraint or drop a table that contains a cross-database constraint, dump both affected databases.
- Simultaneously dump all databases that reference each other. To guard against synchronization problems, put both databases in single-user mode for the dumps. When loading the databases, bring both databases online at the same time.

Cross-database constraints may become inconsistent if you:

- Do not load database dumps in chronological order (for example, you load a dump created on August 12, 1997 after one created on August 13), or
- Load a dump into a database with a new name.

If you do not load the database dumps, cross-database constraints can become inconsistent.

1. Put both databases in single-user mode.
2. Drop the inconsistent referential constraint.
3. Check the data consistency with a query such as:

```
select foreign_key_col from table1
where foreign_key not in
(select primary_key_col from otherdb..othertable)
```

4. Fix any data inconsistency problems.
5. Re-create the constraint.

CHAPTER 15 Restoring the System Databases

The recovery procedure for system databases depends on the database involved and the system problems that necessitated the recovery..

In general, recovery may include:

- Using **load database** to load backups of these databases,
- Using **sybrestore** to restore an SAP ASE database to the time of failure from the most current full database backup dump files.
- Using **dataserver**, **installmaster**, and **installmodel** to restore the initial state of these databases, or
- A combination these tasks.

To make the recovery of system databases as efficient as possible:

- Do not store user databases or any databases other than `master`, `tempdb`, `model`, and `sybsystemdb` on the master device.
- Always keep up-to-date printouts of important system tables.
- Always back up the `master` database after performing actions such as initializing database devices, creating or altering databases, or adding new server logins.

Special procedures are needed because of the central, controlling nature of the `master` database and the master device.

Tables in `master` configure and control all SAP ASE functions, databases, and data devices. The recovery process:

- Rebuilds the master device to its default state when you first installed a server
- Restores the `master` database to the default state
- Restores the `master` database to its condition at the time of your last backup

During the early stages of recovering the `master` database, you cannot use the system stored procedures.

Recovering the master Database

A damaged `master` database can be caused by a media failure in the area on which `master` is stored, or by internal corruption in the database.

Your `master` database is damaged if:

- SAP ASE cannot start.

- There are frequent or debilitating segmentation faults or input/output errors.
- **dbcc** reports damage during a regular check of your databases.

This section describes how to recover the `master` database and to rebuild the master device. It assumes:

- The `master` database is corrupt, or the master device is damaged.
- You have up-to-date printouts of the system tables, listed in *System Administration Guide: Volume 1 > System and Optional Databases*.
- The master device contains only the `master` database, `tempdb`, `model`, and `sybsystemdb`.
- You have an up-to-date backup of the `master` database, and you have not initialized any devices or created or altered any databases since last dumping `master`.
- Your server uses the default sort order.

The *Error Message and Troubleshooting Guide* provides more complete coverage of recovery scenarios.

Recovery Procedure

To restore a damaged master device, follow the recovery steps below, or use the **sybrestore** utility.

You have the option to use the **sybrestore** utility to restore an SAP ASE master database in the event of a master database corruption. See *Master Database Restore* in the *Utility Guide*.

1. Find hard copies of the system tables needed to restore disks, databases and logins.
2. Shut down SAP ASE, and use **dataserver** to build a new `master` database and master device.
3. Restart SAP ASE in master-recover mode.
4. Re-create the `master` database's allocations in `sysusages` exactly.
5. Update the Backup Server network name in the `sysservers` table.
6. Verify that your Backup Server is running.
7. Use **load database** to load the most recent database dump of `master`. SAP ASE stops automatically after successfully loading `master`.
8. Update the **number of devices** configuration parameter in the configuration file.
9. Restart SAP ASE in single-user mode.
10. Verify that the backup of `master` has the latest system tables information.
11. Restart SAP ASE.
12. Check `syslogins` if you have added new logins since the last backup of `master`.
13. Restore the `model` database.

14. Compare hard copies of `sysusages` and `sysdatabases` with the new online version, run **dbcc checkalloc** on each database, and examine the important tables in each database.
15. Dump the `master` database.

Finding Copies of System Tables

Find copies of the system tables that you have saved to a file: `sysdatabases`, `sysdevices`, `sysusages`, `sysloginroles`, and `syslogins`.

Use them at the end of these procedures to guarantee that your system has been fully restored.

See *System Administration Guide: Volume 1 > System and Optional Databases*.

Building a New Master Device

Build a new master device only if your old master device is damaged beyond repair. Otherwise, you can re-create the `master` and `model` databases on your existing master device.

There are two procedures for recreating the master database: replacement of the master device, and forcing SAP ASE to re-create the configuration area. Replace the master device when the master database is corrupted. If the master device's configuration area is also corrupt, force SAP ASE to re-create the configuration area. You can often detect corruption in the configuration area because the server does not run, and produces error messages stating the area is corrupt.

The following examples use the UNIX **dataserver** command. On Windows, use the **sqlsrvr** command.

Replacing the Master Device

You must restart SAP ASE after replacing the master device.

1. Rebuild the master device with the **dataserver -w** option:

   ```
   dataserver -w master
   ```

   Note: Your **dataserver** command may include other options such as command line flags specifying the device path name, server name, interfaces file name, and so on. These options are listed in the `RUN_servername` file that normally starts this server.

   The `-w master` option causes SAP ASE to search the device for database fragments belonging to the `master` database. After it finds these fragments, it writes a default `master` database into that space, then shuts down.
2. Restart SAP ASE with the `RUN_servername` file.

Rebuilding the Configuration Area

If the configuration area is corrupt, you must use the **dataserver -f** option to force SAP ASE to reconstruct the configuration area.

Prerequisites

Note: Your **dataserver** command may include other options such as command line flags specifying the device path name, server name, interfaces file name, and so on. These options are listed in the `RUN_servername` file that normally starts this server.

The following restrictions apply:

- You can specify the page size if it is wrong (for example, `-z8k`).
- You can specify the device size if it is wrong (for example, `-b125M`).
- Any allocation units on the disk that appear corrupt, or that are not currently allocated, are allocated to the `master` database.

Warning! Do *not* attempt to use this procedure to change your server's logical page size. Doing so furthor corrupts the device to the point where you cannot recover it. Any device size you specify must be accurate; **dataserver -w** does not alter the device's size, but it may fail to find parts of some databases if the specified size is too small, and can fail entirely if the specified size is too large.

The `-w master` option, with or without `-f`, re-creates only the master database. All other allocation units on the disk remain unchanged, so you may be able to recover the data using **disk refit**.

If the entire master device is corrupt (for example, in the case of a disk failure), replace the entire device by starting SAP ASE using the **dataserver -z*page_size*. . .-b*device_size***:

Task

1. If the existing master device is not on a raw partition and you plan to reuse it, remove the old master device file.
2. Start the server with **dataserver -z*page_size*. . .-b*device_size***.

Warning! You cannot use this command to change the server's page size at this time. All the other devices for this server use the configured page size, and will not recover properly if you use a different page size. However, since you are creating a new file, you can change the device size.

When determining how large to make your master device, keep in mind that this device reserves 8KB for its configuration area. For example, if you specify the device size as 96MB, some space is wasted because there is not enough space for a full allocation unit. Add an additional .01MB to the space you require for the device to account for this overhead. For example, to use a full 96MB for the device, specify `-b96.01M`.

When you use this method to rebuild the master device, SAP ASE creates new, default databases for `master`, `model`, `tempdb`, and `sybsystemdb`. These databases are all as small as possible for your installation's logical page size. If you intend to load backups to these databases, they may now be too small. Before loading these backups, use **alter database** to increase the size of the databases.

The rebuilt master device is configured with a default character set and sort order. Follow the steps in *System Administration Guide, Volume 1 > Configuring Character Sets, Sort Orders, and Languages* to change the character set and sort order to those used by the original master device.

No matter which method you use to restore your master device, all users and passwords in your master database will be gone. The only remaining privileged login is "sa", with no password.

See the *Utility Guide* for information about re-creating users and passwords.

Starting SAP ASE in Master-Recover Mode

Start SAP ASE in master-recover mode with the **dataserver -m** (UNIX and Windows) option.

- On UNIX platforms – make a copy of the runserver file, naming it `m_RUN_server_`*name*. Edit the new file, adding the parameter `-m` to the **dataserver** command line. Then start the server in master-recover mode:

  ```
  startserver -f m_RUN_server_name
  ```

- On Windows – start SAP ASE from the command line using the **sqlsrver** command. Specify the `-m` parameter in addition to other necessary parameters. For example:

  ```
  sqlsrver.exe -dD:\new_user\DATA\MASTER.dat -sPIANO -eD:\new_user
  \install\errorlog -iD:\new_user\ini -MD:\new_user -m
  ```

See the *Utility Guide* for the complete syntax of these commands.

When you start SAP ASE in master-recover mode, only one login of one user—the system administrator—is allowed. Immediately following a **dataserver** command on the `master` database, only the "sa" account exists, and its password is NULL.

Warning! Some sites have automatic jobs that log in to the server at start-up with the "sa" login. Be sure these are disabled.

Master-recover mode is necessary because the generic `master` database created with **dataserver** does not match the actual situation in SAP ASE. For example, the database does not know about any of your database devices. Any operations on the `master` database could make recovery much more complicated and time-consuming, if not impossible.

An SAP ASE started in master-recover mode is automatically configured to allow direct updates to the system tables. Certain other operations are disallowed.

Warning! Do not make ad hoc changes to system tables—some changes can render SAP ASE unable to run. Make only the changes described in this chapter, and always make the changes in a user-defined transaction.

See also

- *Restarting SAP ASE in Master-Recover Mode* on page 378

Re-creating Device Allocations for master

If you re-created your master device according to the process described in *Rebuilding a New Master Device*, your master database may be too small.

1. From the hard copy version of `sysusages`, total the `size` values shown for `dbid 1` (the `dbid` of the `master` database). Compare those to the size of the current `master` database. You can determine them by issuing:

   ```
   select sum(size)
   from sysusages
   where dbid = 1
   ```

2. If your current master database is too small, use **alter database** to enlarge it to the size you require. To convert logical pages to megabytes, use:

   ```
   select N / (power(2,20) / @@maxpagesize)
   ```

   where *N* is the number of logical pages.

You should not need to alter the size of the master database if you rewrote the master database using the `-m master` option. SAP ASE has recorded the allocation units used by all databases on the device, so you should already have as much space as you need to load your dump of `master`.

Note: If you do not have a hard copy of `sysusages`, you can determine how much larger a database needs to be by attempting to load it. If it is too small, SAP ASE displays an error message telling you how much larger to make it.

Versions of SAP ASE earlier than 15.0 required a complex series of steps to re-create allocations on the new master device. This procedure is no longer necessary. SAP ASE versions 15.0 and later automatically perform most of the work.

Checking Your Backup Server sysservers Information

You must verify that `sysservers` includes the correct information.

Log in to the server as “sa,” using a null password.

1. If the network name of your Backup Server is not SYB_BACKUP, update `sysservers` so that SAP ASE can communicate with its Backup Server. Check the Backup Server name in your interfaces file, and issue:

   ```
   select *
   from sysservers
   where srvname = "SYB_BACKUP"
   ```

2. Check the `srvnetname` in the output from this command. If it matches the interfaces file entry for the Backup Server for your server, go to "Verifying that your Backup Server is running."
3. If the reported `srvnetname` is not the same as the Backup Server in the interfaces file, update `sysservers`. The example below changes the Backup Server network name to PRODUCTION_BSRV:

```
begin transaction
update sysservers
set srvnetname = "PRODUCTION_BSRV"
where srvname = "SYB_BACKUP"
```

4. Execute this command, and verify that it modified only one row. Issue the **select** command again, and verify that the correct row was modified and that it contains the correct value. If **update** modified more than one row, or if it modified the wrong row, issue a **rollback transaction** command, and attempt the **update** again.If the command correctly modified the Backup Server row, issue a **commit transaction** command.

Verifying That Your Backup Server Is Running

On UNIX platforms, use the **showserver** command to verify that Backup Server is running; restart the server if necessary.

See **showserver** and **startserver** in the *Utility Guide*.

SAP Control Center and the Services Manager (on Windows) show whether Backup Server is running.

See the *Utility Guide* for the commands to start Backup Server.

Loading a Backup of master

Load the most recent backup of the `master` database.
For example, on UNIX platforms, use:

```
load database master from "/dev/nrmt4"
```

On Windows, use:

```
load database master from "\\.\TAPE0"
```

After **load database** completes successfully, SAP ASE shuts down. Watch for any error messages during the load and shut down processes.

See also

- *Chapter 14, Backing Up and Restoring User Databases* on page 311

Updating the number of devices Configuration Parameter

You must update the **number of devices** configuration parameter if you use more than the default number of database devices.

Configuration values are unavailable to SAP ASE until after recovery of the `master` database, so instruct SAP ASE to read the appropriate value for the **number of devices** parameter from a configuration file at start-up.

If your most recent configuration file is unavailable, edit a configuration file to reflect the correct value for the **number of devices** parameter.

Edit the runserver file. Add the `-c` parameter to the end of the **dataserver** or **sqlsrver** command, specifying the name and location of the configuration file. When SAP ASE starts, it reads the parameter values from the specified configuration file.

Restarting SAP ASE in Master-Recover Mode

Use **startserver** to restart SAP ASE in master-recover mode.

Watch for error messages during recovery.

Loading the backup of `master` restores the "sa" account to its previous state. It restores the password on the "sa" account, if one exists. If you used **sp_locklogin** to lock this account before the backup was made, the "sa" account is now locked. Perform the rest of the recovery steps using an account with the **sa_role**.

See also

- *Starting SAP ASE in Master-Recover Mode* on page 375

Checking System Tables to Verify Current Backup of master

If you have backed up the `master` database since issuing the most recent **disk init**, **create database**, or **alter database** command, then the contents of `sysusages`, `sysdatabases`, and `sysdevices` match your hard copy.

Check the `sysusages`, `sysdatabases`, and `sysdevices` tables in your recovered server against your hard copy. Look especially for these problems:

- If any devices in your hard copy are not included in the restored `sysdevices`, then you have added devices since your last backup, and you must run **disk reinit** and **disk refit**.
- If any databases listed in your hard copy are not listed in your restored `sysdatabases` table, it means you have added a database since the last time you backed up `master`. You must run **disk refit**.

Note: You must start SAP ASE with trace flag 3608 before you run **disk refit**. However, make sure you read the *Troubleshooting and Error Messages Guide* before you start SAP ASE with any trace flag.

See also

- *Restoring System Tables with disk reinit and disk refit* on page 385

Restarting SAP ASE

Restart SAP ASE in normal (multiuser) mode.

Restoring Server User IDs

Check your hard copy of `syslogins` and your restored `syslogins` table.

- If you have added server logins since the last backup of `master`, reissue the **create login** commands.
- If you have dropped server logins, reissue the **drop login** commands.
- If you have locked server accounts, reissue the **sp_locklogin** commands.
- Check for other differences caused by the use of **alter login** by users or by system administrators.

Make sure that the `suids` assigned to users are correct. Mismatched `suid` values in databases can lead to permission problems, and users may not be able to access tables or run commands.

An effective technique for checking existing `suid` values is to perform a **union** on each `sysusers` table in your user databases. You can include `master` in this procedure, if users have permission to use `master`.

For example:

```
select suid, name from master..sysusers
union
select suid, name from sales..sysusers
union
select suid, name from parts..sysusers
union
select suid, name from accounting..sysusers
```

If your resulting list shows skipped `suid` values in the range where you were restoring logins, add placeholders for the skipped values and then drop them with **drop login** or lock them with **sp_locklogin.**

Restoring the model Database

Restore the `model` database.

1. Load your backup of `model`, if you keep a backup.
2. If you do not have a backup, run the **installmodel** script, which is, on most platforms.

   ```
   cd $SYBASE/$SYBASE_ASE/scripts
   isql -Usa -Ppassword -Sserver_name < installmodel
   ```

   On Windows:

```
cd $SYBASE/$SYBASE_ASE/scripts
isql -Usa -Ppassword -Sserver_name < instmodl
```

3. Redo any changes you made to `model`.

Checking SAP ASE

Compare your hard copy versions of system tables with the versions of the system tables you brought online in the restored databases.

1. Compare your hard copy of `sysusages` with the new online version.
2. Compare your hard copy of `sysdatabases` with the new online version.
3. Run **dbcc checkalloc** on each database.
4. Examine the important tables in each database.

Warning! If you find discrepancies in `sysusages`, call SAP Technical Support.

Backing Up master

When you have completely restored the `master` database and have run full **dbcc** integrity checks, back up the database using your usual dump commands.

Recovering the model Database

Use **dataserver** to restore the `model` database without affecting `master`.

Warning! Shut down SAP ASE before you use any **dataserver** command.

- On UNIX platforms:

  ```
  dataserver -d /devname  -w model
  ```

- On Windows:

  ```
  sqlsrvr -d physicalname -w model
  ```

If you can issue **use model** successfully, you can restore your `model` database from a backup with **load database**.

1. Issue the **dataserver** command described above.
2. If you have changed the size of `model`, reissue **alter database**.
3. Load the backup with **load database**.

If you have changed your `model` database, and you do not have a backup:

1. Issue the **dataserver** command described above.
2. Reissue all the commands you issued to change `model`.

Recovering the sybsystemprocs Database

The `sybsystemprocs` database stores the system procedures that are used to modify and report on system tables.

If your routine **dbcc** checks report damage, and you do not keep a backup of this database, you can restore it using `installmaster`. If you do keep backups of `sybsystemprocs`, you can restore it with **load database.**

Restoring sybsystemprocs with installmaster

If `sybsystemprocs` does not exist, you must create it using **create database**.

These steps assumes that your `sybsystemprocs` database exists, but is corrupt.

1. Check to see which devices currently store `sybsystemprocs`:

   ```
   select  lstart,
   size / (power(2,20)/@@maxpagesize) as 'MB',
   d.name as 'device name',
   case when segmap = 4 then 'log'
   when segmap & 4 = 0 then 'data'
   else 'log and data'
   end as 'usage'
   from sysusages u, sysdevices d
   where d.vdevno = u.vdevno
   and d.status & 2 = 2
   and dbid = db_id('sybsystemprocs')
   order by 1
   ```

   The result probably shows `sybsystemprocs` all on one disk fragment, and having `log and data` as its usage, but you may have a more complex layout than that. Save this query's results for later use.

2. Drop the database:

   ```
   drop database sybsystemprocs
   ```

   If this succeeds and the device is undamaged, go to step 3.

   - If `sybsystemprocs` is badly corrupted, the **drop database** may fail. Manually remove the database by deleting the information that identifies it:

     ```
     sp_configure 'allow updates', 1
     go
     delete from sysusages
     where dbid = db_id('sybsystemprocs')
     delete from sysdatabases
     where name = 'sybsystemprocs'
     go
     sp_configure 'allow updates', 0
     go
     ```

- If the physical disk is damaged, drop the device:

  ```
  sp_dropdevice name_of_sybsystemprocs_device
  ```

If you manually removed `sybsystemprocs`, re-created the `sybsystemprocs` device, shut down SAP ASE using **shutdown with nowait**. If you dropped the `sybsystemprocs` device and it was not a raw partition, remove the physical file. Restart SAP ASE.

3. Re-create the `sybsystemprocs` device. If you dropped the `sybsystemprocs` device, use **disk init** to create a new one. Then re-create `sybsystemprocs` using one of the methods below using the results you saved from step 1.

   Note: If you plan to load a backup copy of `sybsystemprocs`, you can include the **for load** option with the **create database** or **alter database** commands. However, you must use **load database** to load the backup copy before it is used for any other purpose.

   - If the displayed `usage` was all `log and data`, create a simple database using:

     ```
     create database sybsystemprocs on device_name = N
     ```

     where *N* is the total size of the device. You may find that you need to create the new database on multiple devices to get the size you need.
   - If the displayed usage contains any `log` or `data` entries, use **create database** and **alter database** to re-create the same layout. You can group contiguous data or log sections on a single device, but avoid mixing log with data. Re-create the first group of data and log sections:

     ```
     create database sybsystemprocs
     on device_1 = M
     log on device_2 = N
     ```

     where *M* is the sum of the first group of data sizes and *N* is the sum of the first group of log sizes. For each successive group, repeat this process using **alter database** instead of **create database** to enlarge the database.

4. Run the `installmaster` script to create the SAP-supplied system procedures.

   On UNIX platforms:

   ```
   isql -Usa -Ppassword -Sserver_name -i $SYBASE_/$SYBASE_ASE/
   scripts/installmaster
   ```

   On Windows (from the `%SYBASE%\%SYBASE_ASE%\scripts` directory):

   ```
   isql -Usa -Ppassword -S<server_name> -i instmstr
   ```

5. If your site added any procedures or made other changes in `sybsystemprocs`, you must make the same changes in the new `sybsystemprocs` database.

Restoring sybsystemprocs with load database

If you write system procedures and store them in `sybsystemprocs`, you can recover them, when necessary, either by restoring the database, or by loading database backups. .

You can:

- Restore the database from **installmaster**. Then re-create the procedures by reissuing the **create procedure** commands.
- Keep backups of the database, and load them with **load database.**

If you choose to keep a backup of the database, be sure that the complete backup fits on one tape volume or that more than one SAP ASE is can communicate with your Backup Server. If a dump spans more than one tape volume, issue the change-of-volume command using **sp_volchanged**, which is stored in `sybsystemprocs`. You cannot issue that command in the middle of recovering a database.

For example:

- On UNIX, use:

```
load database sybsystemprocs from "/dev/nrmt4"
```

- On Windows, use:

```
load database sybsystemprocs from "\\.\TAPE0"
```

Reducing the Size of tempdb

The `tempdb` (temporary) database provides storage for temporary tables and other temporary working storage needs. If you have a corrupted disk that contains portions of `tempdb`, you should first reduce `tempdb` to its default size and then extend it onto any new device.

Reset tempdb to Default Size

SAP ASE must be in single-user mode to prevent another user from altering the database while you manually update `sysusages`.

Warning! Use this procedure only on `tempdb`. It works because `tempdb` is rebuilt each time the system is shut down and restarted. Using this procedure on any other database results in database corruption.

1. Increase `tempdb` to 4MB on master.
2. Log in to SAP ASE as the system administrator:
3. In case something goes wrong and you need to restore from backup, dump the master database.

   ```
   dump database master to "dump_device"
   ```

   where *dump_device* is the name of the target dump device.

4. To aid in `master` database recovery, if necessary, use the **bcp...out** command to save the following key system tables to data files
   - `master..sysusages`
   - `master..sysdevices`
   - `master..sysdatabases`
   - `master..syslogins`
   - `master..sysconfigures`
   - `master..syscharsets`
   - `master..sysloginroles`
   - `master..sysservers`
   - `master..sysremotelogins`
   - `master..sysresourcelimits`
   - `master..systimeranges`
5. From the `master` database, reconfigure SAP ASE to allow changes to the system catalog:
   ```
   sp_configure "allow updates", 1
   ```
6. Display the current rows belonging to `tempdb` from `sysusages`, and note the number of rows affected:
   ```
   begin transaction
   ```
   ```
   select * from sysusages
   where dbid = db_id('tempdb')
   ```
   The **db_id** function returns the database ID number. In this case, the database ID for `tempdb` is returned.
7. Set the first 2MB of `tempdb` back to data and log in case they were separated:
   ```
   update sysusages
   set segmap = 7 where dbid = db_id('tempdb')
   and lstart = 0
   ```
8. Delete all other rows belonging to `tempdb` from `sysusages`.The number of rows affected should be one less than the number of rows affected by the previous **select** command.
   ```
   delete sysusages where dbid = db_id('tempdb')
   and lstart != 0
   ```

   Warning! Each time SAP ASE is shut down and restarted, the `model` database is copied to `tempdb`. Therefore, if the `model` database has been increased beyond its default size, do not reduce the size of `tempdb` so that it is smaller than `model`.
9. Verify that `tempdb` has one entry that looks like this:
   ```
   select * from sysusages
   where dbid = db_id('tempdb')
   ```
10. If the information is correct, go to step 10 to commit the transaction.

If you see a problem, back out of your changes by entering the following commands:

```
rollback transaction
```

Do not continue with the procedure. Review the steps you performed to determine the cause of the problem.

11. Complete the transaction:

```
commit transaction
```

12. Reconfigure SAP ASE to disallow changes to the system catalog (the normal state for SAP ASE):

```
sp_configure "allow updates", 0
```

13. Immediately issue a **checkpoint** and shut down SAP ASE:

Warning! You must shut down SAP ASE before altering the size of `tempdb` again. If you continue to run without shutting down and restarting, you will receive serious errors on `tempdb`.

```
checkpoint
go
shutdown
go
```

14. Restart SAP ASE.

Restoring System Tables with disk reinit and disk refit

When you are restoring the `master` database from a dump that does not reflect the most recent **disk init** or **create database** and **alter database** commands, restore the proper information in the `sysusages`, `sysdatabases`, and `sysdevices` tables.

See also

- *Checking System Tables to Verify Current Backup of master* on page 378

Restoring sysdevices with disk reinit

If you have added any database devices since the last dump—that is, if you have issued a **disk init** command—you must add each new device to `sysdevices` with **disk reinit**.

If you saved scripts from your original **disk init** commands, use them to determine the parameters for **disk reinit** (including the original value of `vstart`). If the size you provide is too small, or if you use a different `vstart` value, you may corrupt your database.

If you did not save your **disk init** scripts, look at your most recent hard copy of `sysdevices` to determine some of the correct parameters for **disk reinit**. You still need to know the original value of `vstart` if you used a custom `vstart` in the original **disk init** command.

Table 14-1 describes the **disk reinit** parameters and their corresponding `sysdevices` data:

***disk reinit* Parameter**	***sysdevices* Data**	**Notes**
name	`name`	Use the same name, especially if you have any scripts that create or alter databases, or add segments.
physname	*phyname*	Full path to device. Any relative paths are relative to the server's current working directory.
vdevno	**vdevno**	Select a value not already in use.
size	(*high* - *low*) +1	You must provide correct size information.

You can also obtain information on devices by reading the error log for *name*, *physname*, and *vdevno*, and using operating system commands to determine the size of the devices.

If you store your `sybsystemprocs` database on a separate physical device, include a **disk reinit** command for `sybsystemprocs`, if it is not listed in `sysdevices`.

After running **disk reinit**, compare your `sysdevices` table to the copy you made before running **dataserver**.

disk reinit can be run only from the `master` database and only by a system administrator. Permission cannot be transferred to other users.

See *System Administration Guide: Volume 1 > Initializing Database Devices* or the *Reference Manual: Commands*.

Restoring sysusages and sysdatabase with disk refit

If you have added database devices, or created or altered databases since the last database dump, use **disk refit** to rebuild the `sysusages` and `sysdatabases` tables.

disk refit can be run only from the `master` database and only by a system administrator. Permission cannot be transferred to other users. Its syntax is:

```
disk refit
```

SAP ASE shuts down after **disk refit** rebuilds the system tables. Examine the output while **disk refit** runs and during the shutdown process to determine whether any errors occurred.

Warning! Providing inaccurate information in the **disk reinit** command may lead to permanent corruption when you update your data. Check SAP ASE with **dbcc** after running **disk refit**.

CHAPTER 16 Archive Database Access

Archive database access allows a database administrator to validate or selectively recover data from a database dump (an "archive") by making the dump appear as if it were a traditional read-only database; this type of database is called an "archive database."

Unlike a traditional database, an archive database uses the actual database dump as its main disk storage device, with a minimum amount of traditional storage to represent new or modified pages that result from the recovery of the database dump. A database dump already contains the images of many (or even most) of the database pages, therefore, an archive database can be loaded without having to use Backup Server to transfer pages from the archive to traditional database storage. Consequently, the load is significantly faster than a traditional database.

Archive database access lets you perform a variety of operations directly on a database dump.

The amount of storage needed for a traditional database load must be equal to or greater than the size of the source database; the loading of the database dump using Backup Server involves copying pages from the database dump into the storage that has been set aside for the traditional database.

By contrast, you can create an archive database using a minimal amount of traditional disk storage. When you load an archive database, the pages residing in the database dump are not copied by the Backup Server. Instead, SAP ASE creates a map that represents a "logical-to-virtual" mapping of the pages within the archive. This significantly decreases the amount of time required to view the data in a database dump, and reduces the storage requirement for loading the dump.

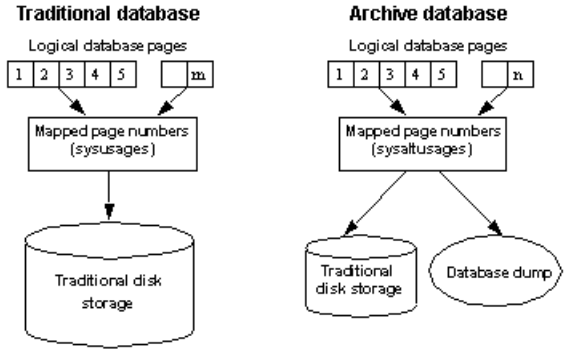
An archive database does not have to be a complete copy of the original database. Depending on the optimization used when dumping the database using **sp_dumpoptimize**, an archive database may be fully populated (every page in the database is in the database dump), or partially populated (only allocated pages are stored in the database dump).

Because the database dump is presented as a read-only database, a database administrator can query it using familiar tools and techniques such as:

- Running database consistency checks on the most recent copy of a dump made from a production database. These checks can be offloaded to a different server to avoid resource contention in the production environment. If resources are not a concern, the archive can be checked on the same server on which it was created. Verification on the archive provides the assurance needed prior to performing a restore operation.
- If the integrity of a database dump is in question, loading it into an archive database can be a quick test for success, and therefore a good tool to identify the appropriate database dump that should be used to restore a traditional database.

- Object-level restoration from the database dump. Lost data is recovered using **select into** to copy the to-be-restored rows from the table within the archive database. Perform the **select into** operation either directly in the server hosting the archive database, or by using Component Integration Services proxy tables if the archive database is available on a different server than that of the object requiring restoration.

In addition, transaction logs can be loaded into an archive database, thereby providing the assurance that the same load sequence can be applied when performing a restore operation. The figure below represents the differences between an archive database and a traditional database structure.



Components of an Archive Database

An archive database is made up of components working together to create the illusion that a database dump is functioning as a traditional database.

These components include:

- The database dump (the archive)
- Optional traditional disk storage used to store the modified pages section
- The scratch database that hosts the `sysaltusages` table

The Database Dump

Archived, read-only database dumps are used as a repository for most unmodified pages.

You cannot make changes to the database dump. Any changes you make to the data within the dump are stored in the modified pages section.

SAP ASE sees the database dump and its stripes as database devices that are usable only by the archive database.

The Modified Pages Section

The archive database that represents a database dump is read-only. No user transactions are allowed, however, some some modifications are allowed.

For example:

- You can run recovery to make the archive database consistent with the source database.
- **dbcc** commands that perform fixes are allowed so that fixed versions of tables can be restored.

These modified and newly allocated database pages cannot be stored within the database dump and its stripes, therefore, an archive database requires some traditional database storage. This disk space is referred to as the modified pages section, and you can allocate it using the **create archive database** and **alter database** commands.

The modified pages section is divided into two segments:

- The disposable changes segment stores any page that is modified or allocated by the recovery undo pass or any log page that is modified or allocated at any time. A page has only one entry in the disposable changes segment.
- The permanent changes segment stores any other page modification or allocation. A page has only one entry in the permanent changes segment.

When you resize the modified pages section, `sysusages` rows are updated in the master database.

The sysaltusages Table and the Scratch Database

`sysaltusages` is a data-only-locked table that maps page numbers in an archive database to the actual page within either the database dump and its stripes, or the modified pages section.

However, unlike the `sysusages` table in a traditional database, `sysaltusages` does not map every logical page in the database, it maps only:

- Pages that have been stored in a database dump
- Pages that have been modified, and therefore, relocated to the modified pages section

See the *Reference Manual: Tables*.

Note: Because `sysaltusages` is a row-locked catalog, you may need to periodically use **reorg** to reclaim logically deleted space.

The scratch database stores the `sysaltusages` table. The scratch database is used to provide flexibility as to where the `sysaltusages` table is located.

The scratch database can be any database (with some exceptions like `master` and `temporary` databases). SAP recommends that you use a dedicated database as the scratch database, because:

- The size of `sysaltusages` may vary, depending on the number of archive databases it supports. You cannot decrease the size of a database, but if it is too large, you can drop it and re-create a smaller database when required.
- It allows you to turn on **trunc log on checkpoint** so that you can automatically truncate the database log.

Apart from hosting the `sysaltusages` table, this database is like any other. You can use threshold procedures and other space management mechanisms to manage space within the database.

To specify a database that is a scratch database, enter

```
sp_dboption <db name>, "scratch database", "true"
```

Each archive database can be assigned to only one scratch database at a time, but multiple archive databases can use the same scratch database. If you have a large number of archive databases, you may want to define multiple scratch databases.

Working With an Archive Database

You can perform many traditional database operations on an archive database.

However, user-defined transactions and commands that modify the database such as such as **insert**, **update**, and **delete** are not allowed.

A populated archive database is similar to a read-only database where the **readonly** option has been applied using **sp_dboption**.

Note: You cannot change the SAP ASE default character set and sort order if it includes an archived database.

DDLGen Support for Archive Database Access

To generate DDL for all archive databases, use the extended filter option “OA.”
For example:

```
ddlgen -Uroy -Proy123 -SHARBAR:1955 -TDB -N% -XOA
```

To generate DDL for a single archive database, use the syntax for normal databases. For example to create DDL for the archive database `archivedb`, enter:

```
ddlgen -Uroy -Proy123 -SHARBAR:1955 -TDB -Narchivedb
```

Configuring an Archive Database

Use **create archive database** to create an archive database.

See the *Reference Manual: Commands*.

Sizing the Modified Pages Section

The modified pages section stores modified or newly allocated database pages.

These pages are stored in either the permanent changes segment, the disposable changes segment, or both.

- A page can be remapped to the permanent changes section only once.
- A page can be remapped to the disposable changes section only once.
- Recovery is responsible for most page remappings.
- **dbcc checkalloc** also requires significant space allocation.
- You can increase the size of the modified pages section using the **alter database** command. However, to decrease the size of the modified pages section, you must drop the archive database and re-create it.

The permanent and disposable changes segments are logically distinct. The permanent changes segment is defined by the set of `sysusages` fragments that contain a `segmap` to the permanent changes segment. The disposable changes segment is defined by the set of `sysusages` fragments containing a `segmap` to the disposable changes segment. The disposable changes segment is discarded at the beginning of each **load tran** command.

Note: SAP ASE automatically manages the division of space in the modified pages section between the permanent changes segment and the disposable changes segment. When this automatic resizing is done, `sysusages` rows are updated in the `master` database.

The minimum size of the modified pages section depends on how many pages are modified or newly allocated in the database. Many of these pages are modified by redo recovery and undo recovery.

You can use the **load database with norecovery** command to minimize the number of modified pages, and therefore, the amount of space required in the modified pages section; however there are downsides to doing this.

Note: dbcc checkalloc consumes a large amount of space in the modified pages section, even if you use the **nofix** option. When you run **dbcc checkalloc**, every allocation page (every 256th page) has information written to it. These allocation-page modifications are stored in the modified pages section, which means that when you are using **dbcc checkalloc,** you need a modified pages section that is at least 1/256th the size of the original database.

If you do not have enough space in the modified pages section, the command that requires the space is suspended and you see an error similar to:

```
There is no more space in the modified pages section for
the archive database <database name>. Use the ALTER
DATABASE command to increase the amount of space
available to the database.
```

To increase space in the modified pages section, either:

- Use **alter database** to increase the size of the modified pages section, or
- If you do not want to allocate additional space to the modified pages section, enter Ctrl+C to abort the current command.

Note: You cannot use thresholds to manage the space in the modified pages section.

Increasing the Amount of Space Allocated to the Modified Pages Section

Use **alter database** to add space to the modified pages section of the archive database.

Increasing the space in the modified pages section allows a suspended command to resume operation.

You can use **alter database** at any time to increase the size of the modified pages section, not only when space runs out.

Materializing an Archive Database

An archive database is a placeholder that is useful only once it has been loaded with a database dump: the load process does not actually copy pages, however, it materializes the database using page mapping.
Use the **load database** command to materialize an archive database.

Note: load database ... norecovery was introduced in SAP ASE version 12.5.4 and 15.0.2 for archive database access. You cannot use **norecovery** on a traditional database. You do not need to have Backup Server running when loading a database dump into an archive database.

Using load database with norecovery

The **with norecovery** option of the **load database** command allows a database dump to be loaded into an archive database without recovering anything, reducing the time required to load.

Many database pages can be modified or allocated during recovery, causing them to be stored in the modified pages section. Therefore, skipping recovery consumes minimum space in the modified pages section. The **with norecovery** option allows a quick view into an archive database.

If you use **with norecovery**, the database is brought online automatically.

However, using **load database with norecovery** for a database that requires recovery may leave it transactionally and physically inconsistent. Running **dbcc** checks on a physically inconsistent database may produce many errors.

Once you have loaded an archive database **with norecovery,** you must have `sa_role` or database owner privileges to use it.

Using Logical Devices with an Archive Database

Use **sp_addumpdevice** to create a logical device from which an archive database can be loaded.

After you have executed **sp_addumpdevice**, use the *logical_name* instead of the *physical_name* as the *dump_device* or *stripe_device* in a **load database** command.

Note: You cannot use an archive database logical device as a device specification for a load into a traditional database or when dumping a traditional database.

load database Limitations with an Archive Database

load database has limitations when used with an archive database.

Including:

- The database dump for an archive database must be a disk dump on a file system mounted on the local machine. This can be local storage or NFS storage. **load database ... at *remote_server*** syntax is not supported, nor are database dumps on tape.
- Cross-architecture loads are not supported. The database dump and the **load database** command must be performed on the same architecture with respect to byte ordering.
- The dumped database must have the same page size as that used by the server that is hosting the archive database.
- The major version of the server on which the dump was taken must be earlier than or equal to the major version of the server hosting the archive database.
- The character set and sort order on the server on which the database dump was taken must be the same as the character set and sort order of the server hosting the archive database.

Bringing an Archive Database Online

Use **online database** to bring an archive database online. **online database** performs undo recovery during which modified and allocated pages may be remapped to the modified pages section.

You need not bring a database online if it has been loaded **with norecovery**, since the load automatically brings the database online without running the recovery undo pass.

Loading a Transaction Log into an Archive Database

When you load a transaction log into an archive database, **load tran** runs the recovery redo pass. Modified and new database pages are written to the permanent changes segment.

You must have enough space in the modified pages section to accommodate these changes. If necessary, increase space for the modified pages section by using **alter database** to increase the normal database storage allocated to the archive database.

Unlike a traditional database, you can bring an archive database online in the middle of a load sequence without breaking the load sequence. When you load a traditional database and then bring it online without using the **for standby_access** clause, you can no longer load the next transaction log in the load sequence. You can, however, bring an archive database online

without the **for standby_access** clause and later load it with the next transaction log in the load sequence. This allows read-only operations like running consistency checks, at any time during the load sequence. When loading a transaction log into the archive database, SAP ASE automatically removes the disposable changes segment from the modified pages section. This effectively reverts the archive database to its state after the previous load was done, thereby allowing the next transaction log in the sequence to be loaded.

Dropping an Archive Database

When dropping an archive database, all the rows for that database are deleted from the `sysaltusages` table in the scratch database. This requires log space in the scratch database.

Use **drop database** to drop an archive database.

SQL Commands for Archive Databases

You can use a number of SQL commands in an archive database.

- **alter database**
- **load database**
- **online database**
- **drop database**
- **load tran**
- **use**
- **select**
- **select into** – where the target database is not an archive database.
- Cursor operations that perform reads, including:
  - **declare cursor**
  - **deallocate cursor**
  - **open**
  - **fetch**

  You cannot use an updatable cursor.
- **checkpoint** – is a supported command. However, the checkpoint process does not automatically checkpoint an archive database.
- **execute** – is allowed as long as any statements that reference the archive database are allowed within the archive database. A transaction inside or outside a stored procedure is not permitted with an **execute** command.
- **lock table**
- **readtext**

Note: DML commands including **insert**, **update**, and **delete** are not permitted, and you cannot start user transactions.

dbcc Commands for Archive Databases

A number of **dbcc** commands are allowed in an archive database.

- **checkdb**
- **checkcatalog**

  Note: The **fix** version of **checkcatalog** is not supported.
- **checktable**
- **checkindex**
- **checkalloc**
- **checkstorage**
- **indexalloc**
- **tablealloc**
- **textalloc**

While **dbcc** commands are executing, other users cannot access an archive database. If you attempt to access an archive database while **dbcc** commands are being performed, you receive a message saying that the database is in single-user mode.

You can use variants of the above **dbcc** commands on an archive database that is online or offline. However, you can use a **dbcc** command with a **fix** option only on an archive database that is online.

Issuing a Typical Archive Database Command Sequence

Generally, creating an archive database includes creating a scratch database, creating the archive database, loading the data, bringing the database online, and loading the transaction log.

1. Create the scratch database, if necessary. To create a 150MB traditional database called `scratchdb`, for example, issue:

   ```
   create database scratchdb
         on datadev1 = 100
         log on logdev1 = 50
   ```

2. Designate the database you just created as a scratch database:

   ```
   sp_dboption "scratchdb", "scratch database", "true"
   ```

3. Create the archive database. This creates an archive database called `archivedb`, with a 20MB modified pages section:

   ```
   create archive database archivedb
         on datadev2 = 20
         with scratch_database = scratchdb
   ```

4. Materialize your archive database using **load database**:

```
load database archivedb
      from "/dev/dumps/050615/proddb_01.dmp"
      stripe on "/dev/dumps/050615/proddb_02.dmp"
```

5. Bring the database online:

```
online database archivedb
```

6. Check the consistency of the archive database:

```
dbcc checkdb(archivedb)
```

7. Load a transaction log dump using **load tran** and restore objects from the archive database using **select into** or **bcp**:

```
load tran archivedb
      from "/dev/dumps/050615/proddb1_log_01.dmp"
load tran archivedb
      from "/dev/dumps/050615/proddb1_log_02.dmp"
online database archivedb
select * into proddb.dbo.orders from       archivedb.dbo.orders
load tran archivedb
      from "/dev/dumps/050615/proddb1_log_03.dmp"
online database archivedb
```

Compressed Dumps for an Archive Database

To use a compressed dump for an archive database, you must create the compressed dump using the **with compression = <compression level>** option of the **dump database** or **dump tran** command, and create a memory pool for accessing the archive database.

Note: Dumps generated with "**compress::**" cannot be loaded into an archive database. Therefore, any references to compression in this chapter refer to dumps generated using the **with compression = <compression level>** option

Creating a Compression Memory Pool

When SAP ASE reads a page from a compressed dump, it selects a compressed block from the dump, decompresses it, and extracts the required page.

The decompression in SAP ASE is done using large buffers from a special memory pool.

Configure the size of the pool using:

```
sp_configure 'compression memory size', size
```

This is a dynamic configuration parameter, and the size is given in 2KB pages. If size is set to 0, no pool is created, and you cannot load a compressed dump.

To determine the optimal size for your pool, consider these two factors:

- The block I/O used by the Backup Server. By default, this block I/O is 64KB, but it could have been changed using the **with blocksize** option in the **dump database** command.

- The number of *concurrent* users decompressing blocks within all archive databases. Each concurrent user requires two buffers, each the same size as the block I/O.

As an absolute minimum, allow one concurrent user two buffers per archive database.

Upgrading and Downgrading an SAP ASE with Archive Databases

You cannot upgrade an archive database. However, you can downgrade an SAP ASE that includes archive databases.

If you load a database dump from an older version of SAP ASE onto an archive database hosted on a newer version of SAP ASE, the database is not internally upgraded when you execute **online database**.

If you upgrade an SAP ASE containing an archive database, all the databases except the archive databases are upgraded. The archive database remains on the older version of SAP ASE.

SAP recommends you reload the archive database with a dump generated from an already upgraded database.

For more information about upgrading SAP ASE, see the installation guide for your platform.

Limitations for Downgrading an SAP ASE with an Archive Database

There are a number of issues you should keep in mind when you downgrade to a version of SAP ASE that does not support archive databases.

- If you must downgrade an SAP ASE containing an archive database to a version of SAP ASE that does not support archive databases, SAP recommends that you drop the archive database before you downgrade.

  To eliminate the new `sysaltusages` table, drop the scratch database before you perform the downgrade procedure. `sysaltuages` does not cause any problems if the scratch database is not dropped.
- Backup Server versions 15.0 ESD #2 and later write a new format for compression (**with compression = *compression_level***) so that the dump can be loaded into an archive database. Therefore, if you must load a compressed dump onto a version of SAP ASE that does not support archive databases access, use the same version of Backup Server to both create and load the compressed database dump. An earlier version of Backup Server does not support the new format of the compressed database dump.

  When you downgrade without compression, you need not worry about Backup Server at all.

Compatibility Issues for a Compressed Dump

Compressed dumps have limitations for archive databases.

- You cannot load dumps generated with "**compress::**" into an archive database. There are no compatibility issues with dumps using this compression option on traditional databases.
- The format of a compressed dump generated using the **with compression = *compression_level*** option has changed in Backup Server 15.0 ESD #2. Therefore:
  - A compressed dump made using a Backup Server version 15.0 ESD #2 and later can be loaded only into a pre-15.0 ESD #2 installation using a Backup Server version 15.0 ESD #2 or later.
  - If you are using a pre-15.0 ESD #2 installation and want to use your dumps for an archive database, use Backup Server version 15.0 ESD #2 or later to create compressed database dumps.

    Note: A Backup Server version 15.0 ESD #2 and later understands both 15.0 ESD #2 and earlier compression formats; therefore, you can use a 15.0 ESD #2 Backup Server for both dumps and loads.

Archive Database Limitations

Using archive databases includes a number of limitations.

- An archive database is read-only.
- Permission to execute commands and stored procedures, and access to objects in an archive database is the same as for a traditional database loaded with the same database dump on the same server.
- When an archive database is loaded **with norecovery**, access to that database is limited to users with `sa_role`, or the database owner.
- You cannot use an in-menory database as an archive database.SAP reommends that you do not use an in-memory database as a scratch database.
- **sybmigrate** does not migrate an archive database if an entire installation is being migrated.
- **sybmigrate** migrates an archive database only if the archive database is specifically selected for migration. When you migrate an archive database to a target server, **sybmigrate** automatically creates a traditional database—rather than an archive database—on the target server.
- An archive database is automatically in single-user mode when any command is run that results in changes to the archive database, such as **dbcc** commands.
- An archive database uses only database dumps or transaction log dumps on disk; tape dumps are not supported.

- The database dump or transaction log dumps must be visible from the server that is hosting the archive database. Remote dumps are not supported.
- For an archive database to access compressed dumps, the dump must have been created using the **with compression** option rather than the "**compress::**" option.
- The checkpoint process does not automatically checkpoint an archive database. Use the **checkpoint** command to checkpoint an archive database.
- You cannot use **sp_dbrecovery_order** to specify an archive database in the database recovery sequence. Archive databases are recovered last, in their *dbid* order.
- When pages are cached in an archive database, the cached pages stay in the memory pool with the same page size as the server. So, on a 2K server, the pages are always cached in a 2K pool. On a 16K server, the pages are always cached in a 16K pool.
- You cannot bind an archive database, or any object within that database, to a user-defined cache. Objects within an archive database default to the default data cache.
- **disk resize** does not work on any device used by an archive database and that maps to a database dump or a transaction log.
- **disk refit** does not rebuild the `master` database's `sysusages` entries from any devices that are used by an archive database. This applies both to dump devices and those used for the modified pages section. Existing `sysusages` entries for an archive database remain however.
- An archive database cannot be replicated.
- An archive database does not fail over on a high-availability server.
- You cannot establish free-space thresholds on an archive database.

CHAPTER 17 Shrinking Databases

SAP ASE allows you to shrink databases, freeing unused space for reuse or deletion.

If SAP ASE encounters data on the portion of the database you are shrinking, that data is moved to a new location before the space is removed from the database. Once the portions to be removed are empty, the physical storage is replaced by references to a null device, making the freed space available for reuse or deletion.

Generally, you can shrink databases that are online and in use. However, in some situations, you must put SAP ASE in single user mode. See "Restrictions."

Note: The shrink database functionality is not supported on the Cluster Edition.

Shrinking a Database

Shrinking a database on a device reduces the amount of free space for all segments on that device.

Use the **alter database** command to shrink databases. The syntax is:

```
alter database database_name
. . .
     off database_device {=size  | [from page_number] [to
page_number]}
     [, database_device…]
     [with timeout='time']
     [with check_only]
```

See *Reference Manual: Commands* for complete syntax and parameter descriptions.

The examples in this section are based the `mydb` database, which is originally configured as:

```
create database mydb on datadev1 = '2G' log on logdev1 = '250M'
alter database mydb on old_dev ='2G'
```

This example shrinks `mydb` to remove the old data (`datadev1` must have sufficient space to receive all data moved from `old_dev`):

```
alter database mydb off old_dev
```

Any data presently on `old_dev` migrates to `new_dev`, and the `mydb` database no longer includes the `old_dev` device. The space `old_dev` occupied is now a data hole (that is, the logical pages exist, but they no longer include storage). If necessary, you can now drop the `old_dev` device.

In this example, `mydb` was extended onto a shared device (in two steps, as the need for more space arose):

```
create database mydb on datadev1 = '2G', shared_dev = '2G' log on
logdev1 = '250M'
alter database mydb on shared_dev = '300M'
alter database mydb on shared_dev = '200M'
```

To remove the 500MB of space that is no longer required:

```
alter database mydb off shared_dev = '500M'
```

The clause ='500M' removes the last 500 megabytes added because that space occupies mydb's highest-numbered logical pages on the shared_dev device, but does not touch the original 2 gigabytes.

This example releases the final 500 megabytes from the first disk piece occupied by shared_dev.

In this example, the data still required resides on the 500 megabytes added to shared_dev, but the original 2G of shared_dev is nearly empty. The server is configured for 8k logical pages, and mydb is described in sysusages as:

```
datadev1 uses logical pages 0 - 262143
shared_dev uses logical pages 262144 - 524287
logdev1 uses logical pages 524288 - 556287
shared_dev uses logical pages 556288 - 594687
shared_dev uses logical pages 594688 - 620287
```

It is much faster to release the empty space than to release space that contains data. To release the final 500 megabytes from the first disk piece occupied by shared_dev, you must determine the page range. Because 500 megabytes on an 8k page represents 64000 logical pages, subtract this value from the start of the disk piece that comes after the one you want to release (in this case logdev1):

```
524288 - 64000 = 460288
```

So, 460288 is the page number for the first page to release.

You must run **alter database ... off** with the database in single user mode to release part of a disk piece.

```
 sp_dboption mydb, 'single user', true
```

See "Restrictions."

To release the final 500 megabytes from the first disk piece occupied by shared_dev:

```
alter database mydb off shared_dev from 460288 to 524288
```

Disable single-user mode:

```
sp_dboption mydb, 'single user', false
```

This example removes the old_dev device (described in example 1), but performs the work 10 minutes at a time, allowing you to use that connection for other work:

```
 alter database mydb off old_dev with time='10:00'
```

Repeat this command until the device is removed (you can perform other work in between iterations).

Note: alter database ... off does not halt after exactly 10 minutes if the command is not finished. Instead, the command works for 10 minutes, then stops at the next stable stopping point (that is, when the allocation unit it is currently working on is released). An allocation unit, a group of 256 pages, is the smallest configurable unit of database storage within SAP ASE.

How SAP ASE Shrinks the Database

For database shrink operations that do not include pages 0 – 255, SAP ASE clears the logical pages and removes them from physical storage. Any data the logical pages contain is moved to other newly allocated pages in the same database.

SAP ASE removes from the databases disk map any cleared data pages that appear at the end of the database. However, if other pages appear in the database after the cleared pages, the cleared pages remain in the database's disk map but point to a null device, which cannot store data.

The resulting disk map may combine both actions—shortening the disk map for some pages, but pointing other pages to a null device.

Note: This version of SAP ASE does not support shrinking databases that include pages 0 - 255.

Shrink Operations on Databases That Contain Text or Image Data

Shrink operations require all text and image columns to have backlinking pointers to the home data row in its first text page.

These pointers are used for several purposes, in general, they are used to locate the home row pointing to the first text page location.

SAP ASE versions 15.7 SP100 and later have the text and image backlinking pointers created and maintained by default. For versions of SAP ASE earlier than 15.7 SP100, use **dbcc shrinkdb_setup** to check a database and mark it for backlink pointer creation.

Replication of the **writetext** command requires access to the home row pointing to the text page where the LOB data is stored. In earlier releases, two methods were used to allow this access: have a back link pointer in the first text page, or use indexes for replication. The process of setting replication at column, table, or database level required an intensive operation to provide the information for the replication support.

Because SAP ASE versions 15.7 SP100 and later have the text and image backlinking pointers created and maintained by default, setting up replication on a new table created in SAP ASE 15.7 SP100 does not require the above intensive operation. SAP ASE ignores the **use_index** parameter of **sp_reptostandby**, **sp_setreptable** and **sp_setrepcol** if the information needed to replicate the LOB columns is already available in the form of back-linked pointers.

When upgrading a database from an earlier version, the back-linked pointers are not available. In this case use **dbcc shrinkdb_setup** at database or object level to update the back-linked pointers. Once the **shrinkdb_setup** operation has completed, the replication indexes will be marked as suspect and will not be used if the replicated object were previously marked to **use_index**. You can use **dbcc reindex** to drop indexes for replication that are no longer needed.

The shrink database operation stops with an error message if it finds text and image data to be moved and the table is not marked to have the backlink pointers created.

Shrink Database Backlink Performance Improvements

Improve performance for backlinking pointers for shrink database operations.

The shrink database operations for 15.7 SP100 and later require that all text and image columns have backlinking pointers to the home data row in the first text page. By default, the text and backlinking pointers are created and maintained if the table contains text, image, or unitext columns. However, this can impact the performance of these commands:

- **create clustered index**
- **reorg rebuild**
- **reorg defrag**
- **alter table** (which involves data copy)
  - unpartition
  - split or move partition
  - drop a column
  - add a NOT NULL column
  - change table scheme from APL to DOL, or vice-versa
  - modify any of these properties of a column
    - the datatype
    - from NULL to NOT NULL, or vice-versa
    - decrease length
    - increase the length of a number column (for example, from tinyint to int)

To improve performance, 15.7 SP121 and later maintains backlinking pointers for the above commands only when tables are set up for replication using text and image backlinking pointers, in which case **sp_reptostandby**, **sp_setreptable**, and **sp_setrepcol** are run without the **use_index** parameter. Otherwise, the tables are marked without text and image backlinking pointers after the command execution.

Important

Since shrink database operations require text and image backlinking pointers be maintained, SAP strongly suggests you follow these steps before shrinking a database:

- Run the proposed shrink database command, adding the qualifier **with check_only**
- Run **dbcc shrinkdb_setup** for each table noted by that command as having its text affected

If you have already initiated a shrink database operation, do not run any of above commands in the database being shrunk, otherwise, the operation can be halted with error 5066:

```
Table <table name> contains off-row columns that do not link back to
their owning rows.
```

Restarting Partially Completed Shrink Operations

Reissuing the same **alter database** command to shrink a database automatically restarts earlier partially completed shrink database operations, resuming the operation from the last completed allocation unit.

Restarting Shrink Operations that Produce Errors

After the shrink database operation moves a page, it updates all the references to this page. However, if the shrink operation runs into changes it could not complete during the page's update, it does not undo the updates it completed. Instead, the shrink operation marks the table `suspect`, restricting it from further use. Restarting the shrink database operation results in error message 5082, and all DML or DDL commands (except **dbcc** commands) that access the table return error message 12347 error when they attempt, but fail, to open the corrupted table.

Use **dbcc dbrepair(dbname, redo_shrink)** to fix any table corruption. This example repairs the `mydb` database:

```
dbcc dbrepair(mydb, redo_shrink)
```

After **dbcc dbrepair** clears the table's `suspect` status, you can restart the shrink database operation, and all DML and DDL commands can access the table.

Sybase recommends that you run **dbcc checktable** after running **dbcc dbrepair** to verify the table is repaired.

Moving Data Before Shrinking the Database

Shrinking a database involves removing physical storage from it without losing the data in that storage.

Each partition in a database must be stored in a segment of the database. The partition is bound to the segment. When SAP ASE moves data during an **alter database ... off** operation, it must move the data to other areas of the database where storage is permitted for this segment.

The **alter database ... off** operation fails if SAP ASE cannot allocate space within the correct segment to receive a page being moved. Any data already moved remains in its new location.

SAP ASE records all data movement in the log record.

Restrictions for Moving the Transaction Log

You cannot move the active portion of the log.

The active portion of the log is any portion of the log allocation unit that has unreplicated transactions, or uncompleted log records for transactions.

Use the **loginfo** function with the **stp_page**, **oldest_transaction_page** and **root_page** parameters to determine the location of these log markers. See the *Reference Manual: Blocks*.

Locks Held During Data Movement

SAP ASE performs data movement one allocation unit at a time, and holds short-duration blocking locks on individual objects.

While processing a given allocation unit, data movement:

1. Obtains the object ID of the next extent to be processed
2. Obtains an exclusive table lock on that object
3. Processes all extents on this allocation unit owned by the locked object
4. Drops the lock

Although the blocking lock stalls other operations on the locked object, the lock is held only briefly. Because shrink database operations must wait for the server to grant exclusive locks, the performance of shrink database operations may suffer when there are concurrent activities on the table.

Determine the Status of a Shrink Operation

Use the **shrinkdb_status** function to determine the status of a shrink operation.

The syntax is:

```
shrinkdb_status(database_name, query)
```

For example, to determine the status of the shrink operation on the pubs2 database, enter:

```
shrinkdb_status("pubs2", "in_progress")
```

Upgrading or Downgrading Shrunken Databases

Upgraded databases may contain replicated tables that are initially setup with **sp_reptostandby 'use index'**, which creates indexes on text and image off-row columns. However, **dbcc shrinkdb_setup** marks these replication indexes as suspect.

You must drop the suspect indexes before you downgrade SAP ASE to an earlier version. Earlier versions of SAP ASE do not recognize suspect text indexes as being unusable, so SAP ASE versions 15.7 SP100 and later cannot downgrade databases containing suspect text indexes. However, the downgraded database may participate in replication since the affected off-row columns contain the correct backlinking information.

Shrink operations create holes (logical pages pointing to null devices that cannot store data) in database disk maps that older versions of SAP ASE do not recognize. The hole is an allocation unit for which there is no associated physical storage. Database holes can occur in data or logs. You cannot downgrade SAP ASE to a version earlier than:

- 15.7 SP100 if it includes databases containing data holes
- 15.7 if it includes databases containing holes of any kind

Fill data holes using **alter database … on**. Fill log holes using **alter database … log on**. See the *Reference Manual: Commands*.

Restrictions

There are several restrictions that apply to shrink operations on databases.

- The shrink database operation may move pages saved as resume points for **reorg** subcommands (for example, **forwarded row**, **reclaim space**, and so on), **reorg** defragment, and similar utilities. The resume points for these utilities become invalid when these pages are moved. You can resume these utilities only at the beginning of the table or partition.
- You cannot shrink master, model, archive, proxy, temporary, inmemory, or other databases with reduced durability.
- You cannot include the **off** parameter with the **alter database ... on**, **log on**, and **log off** parameters.
- You cannot run a shrink database operation simultaneously with **dump database**, **dump transaction**, or other **alter database** commands in a single database. The shrink database operation cannot run if these commands are in progress in the indicated database.

  Shrink database operations fail if you start them while a **dump transaction** is already running. If you issue **dump transaction** while a shrink database operation is in progress, SAP ASE terminates the shrink database operation to allow the **dump transaction** to proceed. You can restart the shrink operation by issuing the same command after the **dump transaction** completes.

- You must run a shrink database operation in single user mode if you specify a page range to shrink only part of a device.
- Shrink operations may create a hole when they remove space from the middle or end of a database. You can perform transaction log dumps after a shrink database operation that creates a hole in the data segment, although the transaction log may include a dependency on the data that occupied the hole when you load the transaction log.

  SAP ASE may issue error number 3193 and this message when you load the transaction log: "`Database '<dbname>' does not have space allocated for data page <pageid> from the dump. Extend the database by <size> MB and retry the command`"..

  You may need to extend the database before some (and potentially, every) subsequent transaction log dump in the load sequence.

  To avoid this error pro-actively, Sybase recommends that you perform a database dump immediately after you perform a shrink database operation.

  To load a sequence of transaction log dumps that follow a shrink operation that did not include a subsequent database dump:

  - Run **load transaction**
  - If you cannot run **load transaction** because SAP ASE issues error number 3193:
    1. Query `sysusages` to determine which holes require filling.
    2. Run **alter database** to fill the holes, extending the database to avoid error number 3193
    3. Run **load transaction** using the same transaction log dump

  Note: During these sequence of steps, the data holes you filled with the **alter database** command may be re-created by the **load transaction** command, and SAP ASE may report error number 3193 error when you perform the next **load transaction**. If this occurs, repeat these steps for every **load transaction** you perform.
- Shrink database operations do not update indexes marked as `suspect` on tables whose pages will be moved as part of the shrink operation. After the shrink operation completes, these indexes remain marked as `suspect`, and may include references to data pages that are part of a device hole. Running **dbcc checktable** on tables with `suspect` indexes may result in error number 806 (this is expected). Instead, drop and re-create these indexes, or repair them with **reorg rebuild table** or **dbcc reindex(table_name, 16)**.

CHAPTER 18 Expanding Databases Automatically

The automatic database expansion stored procedure **sp_dbextend** allows you to install thresholds that identify devices with available space, and then appropriately alter the database —and the segment where the threshold was fired—on these devices.

After you set up a database for automatic expansion, internal mechanisms fire when a database grows to its free space threshold, and increase the size of the database by the amount of space your expansion policies specify. The automatic expansion process measures the amount of room left on all devices bound to the database. If there is sufficient room on the devices, the database continues to grow. By default, if the size of the device is greater than 40MB, the size of the database is increased by 10 percent. If your database is smaller than 40MB, the size of the database is increased by 4MB. However, you can specify database resizing limits that match the needs of your site.

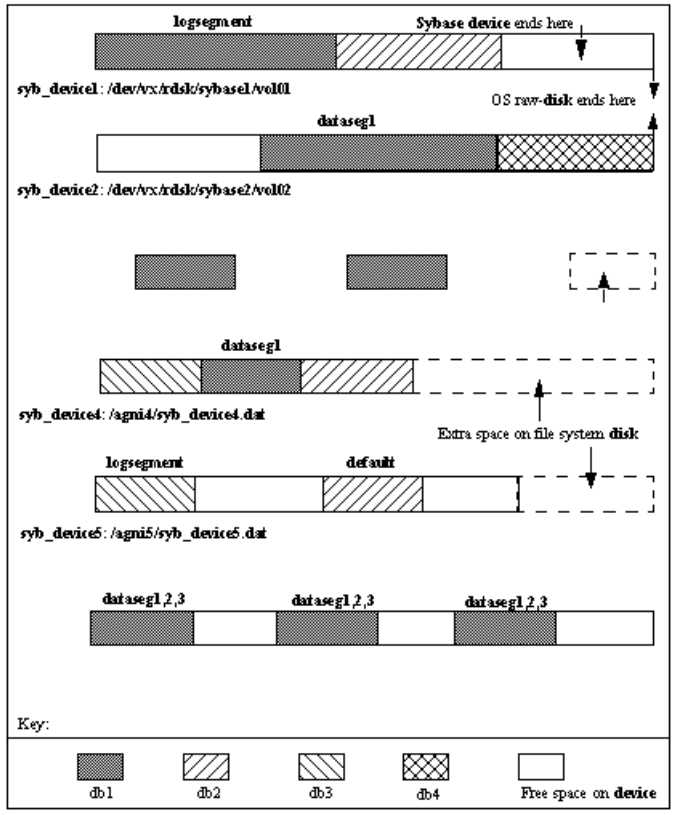
If any devices are configured for expansion, those devices expand next. Finally, the database is expanded on those devices.

This automatic expansion process runs as a background task and generates informational messages in the server's error log.

Layouts for Disks, Devices, Databases, and Segments

Raw disks can be partially occupied by a SAP device. Configure a layout for you site's physical resources that makes the best use of space.

This figure shows the various layouts of physical resources that may exist in an SAP ASE installation after a series of **disk init**, **create database,** and **alter database** operations. You can use this information to devise different plans of physical and logical space layout while testing stored procedures.



`syb_device2` shows an entire raw disk fully occupied by a single SAP device, on which multiple databases were created. On this raw disk (`/dev/vx/rdsk/sybase2/vol02`), there is still some empty space at the head of the device, which can happen when a database that initially occupied this space is subsequently dropped.

`syb_device4` and `syb_device5` show the layout of SAP devices `/agni4/syb_device4.dat` and `/agni5/syb_device5.dat` on a file system disk, where the SAP device occupies a portion of the disk, but there is still room for the device (an operating system file, for instance) to grow.

`syb_device6` shows a SAP file system disk that has fully occupied the entire available space on the physical disk, but still has unused space on the device. This space can be used to expand existing databases on this device.

These various devices illustrate database fragments for different databases. Each device for a particular database has one or more segments spanning that device.

In `syb_device6`, `/agni6/syb_device6.dat`, `db1` spans three separate pieces of the device. That device also belongs to three different segments, data segments 1, 2, and 3. All

three entries in `sysusages` for this database, `db1`, appear with a segment map value that includes all three segments.

However, on the device `syb_device3` on `/dev/raw/raw3`, the database has two pieces of the device, and one of them is exclusively marked as for the log segment while the other is marked as for the default segment. Assuming that an initial **create database** command has already been executed, the following SQL commands can produce this result:

```
alter database db1 on syb_device3 = "30M"
alter database db1 log on syb_device3 = "10M" with       override
```

The first **alter database** command creates the database piece of default segment, and the second one creates the database piece of `logsegment`, forcing an override even though both pieces are on the same device. Space is expanded on a database in individual segments.

Threshold Action Procedures

Database expansion is performed by a set of threshold action procedures that are fired when free space crosses a threshold that is set for a segment. **sp_dbextend** is the interface for administering and managing the expansion process for a specified segment or device.

You can configure automatic expansion to run with server-wide default expansion policies, or you can customize it for individual segments in specified databases.You can install thresholds on key segments on which tables with critical data reside, allowing you a fine degree of control over how SAP ASE meets the data space requirements for different kinds of tables. If your site has key tables with large volumes of inserts, you can bind these tables to specific segments, with site-specific rules for extending that segment. This enables you to avoid outages that can occur in a production environment with large loads on such key tables.

You cannot use the thresholds to shrink a database or its segment.

See the *Reference Manual: Procedures*.

Installing Automatic Database Expansion Procedures

Install automatic expansion using the `installdbextend` script, which loads rows into `master.dbo.sysattributes`, which describes defaults for automatic expansion in a database or in a device.

The installation script also creates a control table in the `model` and `tempdb` databases.

If you are upgrading to SAP ASE version 12.5.1 or later, you must install this script separately as part of your upgrade process.

1. Log in with **sa_role** permissions. In UNIX, `installdbextend` is located in `$SYBASE/$SYBASE_ASE/scripts`. If you are running Windows, the location is `%SYBASE%/%SYABASE_ASE%/scripts`
2. On UNIX, run:

   ```
   isql -Usa -P -Sserver_name <$SYBASE/$SYBASE_ASE/scripts/
   installdbextend
   ```

   On Windows, run:

   ```
   isql -Usa -P -Sserver_name <%SYBASE%\%SYBASE_ASE%\scripts
   \installdbextend
   ```

   `installdbextend` script installs the family of threshold action procedures and **sp_dbextend** in the `sybsystemprocs` database.

Running sp_dbextend

sp_dbextend allows you to customize the database and device expansion process, based on site-specific rules.
Use this syntax for **sp_dbextend**:

```
sp_dbextend [ command [, arguments...] ]
```

where **command** is one of the options discussed below, and **arguments** specifies the database name, segment name, amount of free space, and so on. See the *Reference Manual: Procedures*.

Database administrators can configure the size or percentage by which each database and segment pair, and device should be expanded.

You can also limit the expansion of a database segment or device by specifying a maximum size beyond which no further expansion is possible. Use **sp_dbexpand** in a test mode to simulate the expansion processes based on your chosen policies.

sp_dbextend provides ways to list the current settings and drop site-specific policy rules.

This information is stored as new attribute definitions in `master.dbo.sysattributes`.

If you do not include an argument, **sp_dbextend** defaults to **help**. See the *Reference Manual: Procedures*.

Note: The automatic expansion procedure does not create new devices; it only alters the size of the database and segment on existing devices to which the segment currently maps.

To discontinue threshold action procedure, clear the threshold using the **sp_dropthreshold**, or use **sp_dbextend** with the *clear* option. See the *Reference Manual: Procedures*.

Validating Current Thresholds

Use the **check** parameter to validate the current settings of various thresholds.

For instance, **check** warns you if multiple segments share the same set of devices and both segments are set for automatic expansion, or if the threshold currently set to trigger automatic expansion on the `logsegment` is too close to the current last-chance threshold for the `logsegment`. In this situation, the automatic threshold does not fire, and **check** reports a warning.

sp_dbextend includes a powerful simulation mode that any user with **sa_role** permission can use to simulate the execution of the top-level threshold action procedure.

- To define expansion policies for the `logsegment` in the `pubs2` database:

```
sp_dbextend 'set', 'database', pubs2, logsegment,'3M'
sp_dbextend 'set', 'threshold', pubs2, logsegment, '1M'
```

- To simulate expansion for these policies:

```
sp_dbextend 'simulate', pubs2, logsegment
------------------------------
```

Messages from the server follow this input.
The following examples show the series of database and disk expansions that would occur if the threshold on database `pubs2` segment `logsegment` fired once:

```
sp_dbextend 'simulate', pubs2, logsegment
---------------
NO REAL WORK WILL BE DONE.
Simulate database / device expansion in a dry-run mode 1 time(s).
These are the series of database/device expansions that would have
happened if the threshold on database'pubs2',
segment 'logsegment' were to fire 1 time(s).

Threshold fires: Iteration: 1.
==============================

Threshold action procedure 'sp_dbxt_extend_db' fired in db 'pubs2'
on segment 'logsegment'.

Space left: 512 logical pages ('1M').ALTER DATABASE pubs2 log on
pubs2_data = '3.0M'
-- Segment: logsegmentDatabase 'pubs2' was altered by total size
'3M' for
segment 'logsegment'.
Summary of device/database sizes after 1 simulated extensions:
================================================
devicename initial size final size
---------- ------------ ----------
pubs2_data 20.0M        20.0M

(1 row affected)

Database 'pubs2', segment 'logsegment' would be altered from an
initial
```

```
size of '4M' by '3M' for a resultant total size of '7M'.

To actually expand the database manually for this threshold,
issue:
sp_dbextend 'execute', 'pubs2','logsegment', '1'

(return status = 0)
```

To expand the database manually for this threshold, execute:

```
sp_dbextend ’execute’, ’pubs2’, ’logsegment’
--------------
```

This example shows that if the threshold fires at this level, an **alter database** command operates on the `pubs2_data` device for the `logsegment`:

```
sp_dbextend 'execute', pubs2, logsegmentThreshold fires:
Iteration: 1.
================================
Threshold action procedure 'sp_dbxt_extend_db' fired in db 'pubs2'
on
segment 'logsegment'. Space left: 512 logical pages ('1M').

ALTER DATABASE pubs2 log on pubs2_data = '3.0M' -- Segment:
logsegment
Extending database by 1536 pages (3.0 megabytes) on disk
pubs2_data
Warning: The database 'pubs2' is using an unsafe virtual device
'pubs2_data'. The recovery of this database can not be guaranteed.

Warning: Using ALTER DATABASE to extend the log segment will cause
user
thresholds on the log segment within 128 pages of the last chance
threshold to be disabled.

Database 'pubs2' was altered by total size '3M' for segment
'logsegment'.

(return status = 0)
```

- To simulate what would actually happen if the threshold fired *<n>* times in succession on a particular segment, issue the same command, specifying the number of iterations:

  ```
  sp_dbextend ’simulate’, pubs2, logsegment, 5
  -----------------
  ```

  To expand this database five times, enter:

  ```
  sp_dbextend ’execute’, ’pubs2’, ’logsegment’, 5
  ------------------
  ```

  If you execute this command, you see in the output that firing the threshold five times in succession puts the database through a series of **alter database** operations, followed by one or more **disk resize** operations and, finally, an **alter database** on the specified device.

Configuring a Database for Automatic Expansion

You can set up different segments in a database for automatic expansion.

The example below uses the `pubs2` database. Not all these steps are mandatory. For example, you may choose not to set *growby* or *maxsize* for individual devices, and to use the system default policies only for the devices.

1. Create the database:

```
 create database pubs2 on pubs2_data = "10m" log on pubs2_log =
"5m"
```

2. Set the *growby* and *maxsize* policies for the `pubs2_data` device at 10MB and 512MB respectively. You can be in any database to set these policies. Enter:

```
 exec sp_dbextend 'set', 'device', pubs2_data, '10m', '512m'
```

3. The system default *growby* policy is 10% for devices. Rather than set new policies for the `pubs2_log` device, you can modify this system default, choosing an appropriate *growby* value. The `pubs2_log` then expands at this rate. Enter:

```
exec sp_dbextend 'modify', 'device', 'default', 'growby', '3m'
```

4. Set the *growby* rate for the default segment, but do not specify a maximum size. Enter:

```
 exec sp_dbextend 'set', 'database', pubs2, 'default', '5m'
```

   The *growby* rate on the default segment may be different from that on the devices where the segment resides. *growby* controls the segment's expansion rate when it is running out of free space, and is used only when you expand the segment.

5. Set the *growby* and *maxsize* variables for the `logsegment`:

```
 exec sp_dbextend 'set', 'database', pubs2, 'logsegment', '4m',
'100m'
```

6. Examine the policies established for various segments in the `pubs2` database:

```
exec sp_dbextend 'list', 'database', pubs2
```

7. Examine the policies in the various devices that `pubs2` spans. The pattern specifier for *devicename* ("%") picks up all these devices:

```
exec sp_dbextend 'list', 'device', "pubs2%"
```

8. Install the expansion threshold for the default and `logsegments` segments in `pubs2`. This sets up and enables the expansion process, and allows you to choose the free space threshold at which to trigger the expansion process. Enter:

```
 use pubs2
------------------------------------------------------------------
   exec sp_dbextend 'set', 'threshold', pubs2, 'default',   '4m'
    exec sp_dbextend   'set', 'threshold',  pubs2,  'logsegment',
'3m'
```

9. Examine the thresholds installed by the commands above.

```
exec sp_dbextend list, 'threshold'

segment name free pages  free pages (KB) threshold procedure
status
------------ ----------- --------------- -------------------
-----------
default          2048    4096            sp_dbxt_extend_db enabled
logsegment        160  320           sp_thresholdaction lastchance
logsegment       1536  3072          sp_dbxt_extend_db      enabled
Log segment free space currently is 2548 logical pages (5096K).
(1 row affected, return status = 0)
```

   In this output, **sp_dbxt_extend_db** is the threshold procedure that drives the expansion process at runtime. The expansion thresholds are currently enabled on both the default and `logsegment` segments.

10. Use **simulate** to see the expansion:

```
exec sp_dbextend 'simulate', pubs2, logsegment
exec sp_dbextend 'simulate', pubs2, 'default', '5'
```

11. Use **modify** to change the policy, if necessary:

```
exec sp_dbextend 'modify', 'database', pubs2, logsegment,
'growby','10m'
```

12. To disable expansion temporarily on a particular segment, use **disable**:

```
 exec sp_dbextend 'disable', 'database', pubs2, logsegment
```

13. Examine the state of the expansion policy on databases and devices:

```
exec sp_dbextend list, 'database'
name         segment    item    value status
------------ ---------- ------- ----- --------
server-wide (n/a)       (n/a)   (n/a) enabled
default     (all)       growby  10%   enabled
pubs2       default     growby  5m    enabled
pubs2       logsegment  growby  10m   disabled
pubs2       logsegment  maxsize 100m  disabled
(1 row affected, return status = 0)
```

   The status disabled indicates that the expansion process is currently disabled on the `logsegment` in `pubs2`.

```
exec sp_dbextend list, 'device'
name           segment item   value status
-------------- ------- ------ ----- -------
server-wide    (n/a)   (n/a)  (n/a) enabled
default        (n/a)   growby 3m    enabled
mypubs2_data_0 (n/a)   growby 10m   enabled
mypubs2_data_1 (n/a)   growby 100m  enabled
mypubs2_log_1  (n/a)   growby 20m   enabled
mypubs2_log_2  (n/a)   growby 30m   enabled
(1 row affected, return status = 0)
```

14. Use **enable** to reenable the expansion process:

```
exec sp_dbextend 'enable', 'database', pubs2, logsegment
```

Restrictions and Limitations

Setting thresholds includes certain restrictions and limitations.

- When threshold procedures are installed on multiple segments in one or more databases, the expansion is performed in the order in which the thresholds fire. If **abort tran on log full** is off for the `logsegment`, tasks wait until the threshold procedure for the `logsegment` is scheduled to alter the database.
- In unlogged segments, tasks continue to process even after the free space threshold is crossed, while the threshold procedure remains in queue. This can cause "out of space" errors in data segments. Design your thresholds to have sufficient space in the database for the task to complete.
- If many threshold procedures fire simultaneously, the procedure cache may become overloaded. This is more likely to occur in installations with large numbers of databases, many segments, and many threshold action procedures installed.
- If the space in the `tempdb` is very low, and other operations need `tempdb` resources, the threshold procedures may fail even while trying to correct the situation. Make sure that threshold procedures in `tempdb` are installed with sufficiently large amounts of free space, at least 2MB, to avoid this problem.

You may need to change your dump and load procedures to manage site-specific policies that determine how databases and devices expand.

Dumping a database does not transport information stored in `master.dbo.sysattributes`, so if you use dump and load to migrate databases from a source server to a target server, you must manually migrate any site-specific policies encoded as data in the `sysattributes` database. There are two possible workarounds:

- Using **bcp out** from a view defined on `master.dbo.sysattributes` for entries with class number 19, you can manually extract the data from `master.dbo.sysattributes`, then use **bcp in** to load the data into the target server. This requires that both databases across the two servers have the same segment IDs.
- You can also use the **ddlgen** feature of SAP Command Center to regenerate the **sp_dbextend set** invocations necessary to re-create your policy rules, by running the **ddlgen** script at the target server. However, you cannot use **ddlgen** to manage renamed logical devices across servers procedure. You must manually rename the devices at the target server.

These restrictions do not cause failure:

- You can install a threshold action on an unlogged segment when the database has **sp_dboption 'no free space acctg'** enabled (see the *Reference Manual: Procedures*). This

option means only that no database expansion is performed, since threshold actions are not fired with this option is off. Leaving this option on generates a warning message.

- SAP also recommends that you periodically dump the `master` database if expansion occurs, so that you can re-create the `master` database in case of failure after several expansions.
- SAP recommends that you do not install these generic threshold procedures on any system databases, particularly the `master` database, as modifying space usage in the `master` database requires special treatment (see the *Reference Manual: Commands*.)
- You cannot use thresholds to shrink a database or segment.

CHAPTER 19 Managing Free Space with Thresholds

When you create or alter a database, you allocate a finite amount of space for its data and log segments. As you create objects and insert data, the amount of free space in the database decreases.

See also

- *Placing a Transaction Log on a Separate Device* on page 121
- *Querying the syslogshold Table* on page 449

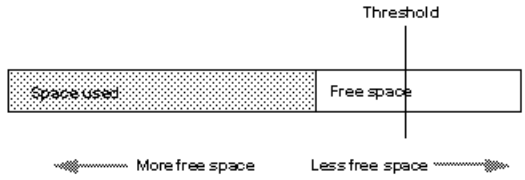
Monitoring Free Space with the Last-Chance Threshold

All databases, including `master`, have a *last-chance threshold*, which is an estimate of the number of free log pages that are required to back up the transaction log. As you allocate more space to the log segment, SAP ASE automatically adjusts the last-chance threshold.

When the amount of free space in the log segment falls below the last-chance threshold, SAP ASE automatically executes a special stored procedure called **sp_thresholdaction**. (Use **sp_modifythreshold** to specify a different last-chance threshold procedure. See the *Reference Manual: Procedures*).

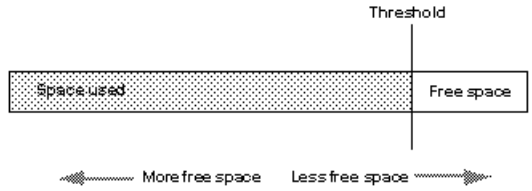
This figure illustrates a log segment with a last-chance threshold. The shaded area represents used log space; the unshaded area represents free log space. The last-chance threshold has not yet been crossed.

Figure 26: Log Segment with a Last-Chance Threshold



As users execute transactions, the amount of free log space decreases. When the amount of free space crosses the last-chance threshold, SAP ASE executes **sp_thresholdaction**.

Figure 27: Executing sp_thresholdaction when the Last-Chance Threshold Is Reached



Controlling How Often sp_thresholdaction Executes

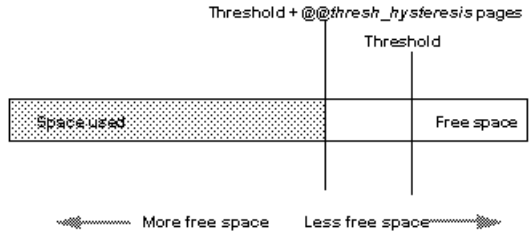
SAP ASE uses a hysteresis value, the global variable *@@thresh_hysteresis*, to control the sensitivety levels of thresholds to variations in free space.

A threshold is deactivated after it executes its procedure, and remains inactive until the amount of free space in the segment rises *@@thresh_hysteresis* pages above the threshold. This prevents thresholds from executing their procedures repeatedly in response to minor fluctuations in free space.

You cannot change the value of *@@thresh_hysteresis*.

For example, when the threshold in the previous figure executes **sp_thresholdaction**, it is deactivated.

In the following figure, the threshold is reactivated when the amount of free space increases by the value of *@@thresh_hysteresis*:



Rollback Records and the Last-Chance Threshold

Rollback records are logged whenever a transaction is rolled back. Servers save enough space to log a rollback record for every update belonging to an open transaction.

If a transaction completes successfully, no rollback records are logged, and the space reserved for them is released.

In long-running transactions, rollback records can reserve large amounts of space.

To check the space used by the `syslogs`, run **sp_spaceused**:

```
sp_spaceused syslogs

name            total_pages     free_pages      used_pages
reserved_pages
--------------- --------------- --------------- --------------
--------------
syslogs               5632           1179           3783           670
```

dbcc checktable(syslogs) produces similar output:

```
Checking syslogs: Logical pagesize is 2048 bytes
The total number of data pages in this table is 3761.
*** NOTICE: Space used on the log segment is 3783 pages (7.39
Mbytes), 67.17%.
*** NOTICE: Space reserved on the log segment is 670 pages (1.31
Mbytes), 11.90%.
*** NOTICE: Space free on the log segment is 1179 pages (2.30
Mbytes), 20.93%.
```

If the last chance threshold for the transaction log fires when it seems to have sufficient space, it may be the space reserved for rollbacks that is causing the problem.

See also

- *Determining the Current Space for Rollback Records* on page 422

Calculating the Space for Rollback Records

Add space to the transaction log to accomodate rollback records.

To calculate the amount of space to add to a transaction log for rollback records, estimate:

- The number of update records in the transaction log that are likely to belong to transactions that have already rolled back
- The maximum number of update records in the transaction log that are likely to belong to open transactions at any one time

Update records change the timestamp value, and include changes to data pages, index pages, allocation pages, and so on.

Each rollback record requires approximately 60 bytes of space, or 3/100ths of a page. Thus, the calculation for including rollback records (RRs) in the transaction log is:

Added space, in pages = (logged RRs + # open updates) X 3/100

You may also want to add log space to compensate for the effects of rollback records on the last-chance threshold and on user-defined thresholds, as described in the following sections.

Using lct_admin to Determine the Free Log Space

Use **logsegment_freepages** to determine the amount of free space your dedicated log segment has.

To see the number of free pages for the `pubs2` database log segment, enter:

```
select lct_admin("logsegment_freepages", 4)
---------------------
                   79
```

Determining the Current Space for Rollback Records

To determine the number of pages a database currently reserves for rollbacks, issue **lct_admin** with the **reserved_for_rollbacks** parameter.

The number of pages returned are the number reserved, but not yet allocated, for rollback records.

For example, to determine the number of pages reserved for rollbacks in the `pubs2` database (which has a `dbid` of 5), issue:

```
select lct_admin("reserved_for_rollbacks", 5)
```

See the *Reference Manual: Commands.*

See also

- *Rollback Records and the Last-Chance Threshold* on page 420

Effect of Rollback Records on the Last-Chance Threshold

SAP ASEs that use rollback records must reserve additional room for the last-chance threshold.

The last-chance threshold is also likely to be reached sooner because of the space used by already logged rollback records and the space reserved against open transactions for potential rollback records.

User-Defined Thresholds

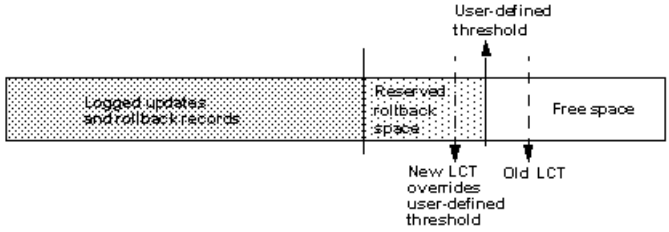
Because rollback records occupy extra space in the transaction log, there is less free space after the user-defined threshold for completing a dump than in versions of SAP ASE that do not use rollback records.

However, the loss of space for a dump because of the increased last-chance threshold is likely to be more than compensated for by the space reserved for rollback records for open transactions.

You can use a user-defined threshold to initiate a **dump transaction**. Set the threshold so there is enough room to complete the dump before the last-chance threshold is reached and all open transactions in the log are suspended.

In databases that use mixed log and data, the last-chance threshold moves dynamically, and its value can be automatically configured to be less than the user-defined threshold. If this happens, the user-defined threshold is disabled, and the last chance threshold fires before the user-defined threshold is reached:

Figure 28: LCT Firing Before User-Defined Threshold



The user-defined threshold is reenabled if you set the value for the last-chance threshold greater than the user-defined threshold (for example, if the last chance threshold is reconfigured for the value of “Old LCT” in the previous image).

In databases that have a separate log segment, the log has a dedicated amount of space and the last-chance threshold is static. The user-defined threshold is not affected by the last-chance threshold.

Last-Chance Threshold and User Log Caches for Shared Log and Data Segments

Every database in an SAP ASE has a last-chance threshold, and all databases allow transactions to be buffered in a user log cache.

When you create a database with shared log and data segments, its last-chance threshold is based on the size of the `model` database. As soon as data is added and logging activity begins, the last-chance threshold is dynamically recalculated, based on available space and currently open transactions. The last-chance threshold of a database with separate log and data segments is based on the size of the log segment and does not vary dynamically.

To get the current last-chance threshold of any database, use **lct_admin** with the **reserve** parameter and a specification of 0 log pages:

```
select lct_admin("reserve",0)
```

The last-chance threshold for a database is stored in the `systhresholds` table and is also accessible through **sp_helpthreshold**. However:

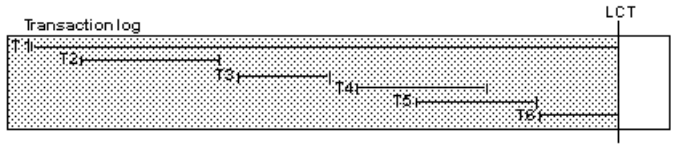
- **sp_helpthreshold** returns user-defined thresholds and other data, as well as an up-to-date value for the last-chance threshold. Using **lct_admin** is simpler if you need only the current last-chance threshold. Either of these values produce the most current value for the last-chance threshold.
- For a database with shared log and data segments, the last-chance threshold value in `systhresholds` may not be the current last-chance threshold value.

Using lct_admin abort to Abort Suspended Transactions

All open transactions are suspended when the transaction log reaches the last-chance threshold.

Typically, space is created by dumping the transaction log, since this removes committed transactions from the beginning of the log. However, if one or more transactions at the beginning of the log is still open, it prevents a dump of the transaction log.

Use **lct_admin abort** to terminate suspended transactions that are preventing a transaction log dump. Since terminating a transaction closes it, this allows the dump to proceed. This figure illustrates a possible scenario for using **lct_admin abort**. A transaction log has reached its LCT, and open transactions T1 and T6 are suspended. Because T1 is at the beginning of the log, it prevents a dump from removing closed transactions T2 through T5 and creating space for continued logging. Terminating T1 with **lct_admin abort** allows you to close T1 so that a dump can clear transactions T1 through T5 from the log



Before you can abort a transaction, you must first determine its ID.

1. Use the following query to find the `spid` of the oldest open transaction in a transaction log that has reached its last-chance threshold:

   ```
   use master
   go
   select dbid, spid from syslogshold
   where dbid = db_id("name_of_database")
   ```

   For example, to find the oldest running transaction on the `pubs2` database:

   ```
   select dbid, spid from syslogshold
   where dbid = db_id ("pubs2")
   dbid   spid
   ------ ------
       7      1
   ```

2. To terminate the oldest transaction, enter the process ID (`spid`) of the process that initiated the transaction, which also terminates any other suspended transactions in the log that belong to the specified process.

   For example, if process 83 holds the oldest open transaction in a suspended log, and you want to terminate the transaction, enter:

   ```
   select lct_admin("abort", 83)
   ```

This also terminates any other open transactions belonging to process 83 in the same transaction log.

To terminate all open transactions in the log, enter:

```
select lct_admin("abort", 0, 12)
```

See the *Reference Manual: Commands.*

Add Space to the Master Database's Transaction Log

When the last-chance threshold on the `master` database is reached, you can use **alter database** to add space to the `master` database's transaction log.

This allows more activity in the server by activating suspended transactions in the log. However, while the master transaction log is at its last-chance threshold, you cannot use **alter database** to make changes in other databases. Thus, if both `master` and another database reach their last-chance thresholds, you must first use **alter database** to add log space to the `master` database, and then use it again to add log space to the second database.

Automatically Aborting or Suspending Processes

By design, the last-chance threshold allows enough free log space to record a **dump transaction** command. There may not be enough room to record additional user transactions against the database.

When the last-chance threshold is crossed, SAP ASE suspends user processes and displays the message:

```
Space available in the log segment has fallen critically low in
database 'mydb'. All future modifications to this database will be
suspended until the log is successfully dumped and space becomes
available.
```

You can now execute only commands that are not recorded in the transaction log (**select** or **readtext**) and commands that might be necessary to free additional log space (**dump transaction**, **dump database**, and **alter database**).

Using abort tran on log full to Abort Transactions

Use **sp_dbotion** to configure the last-chance threshold to automatically abort open transactions, rather than suspend them.
The syntax is:

```
sp_dboption database_name "abort tran on log full", true
```

If you upgrade from an earlier version of SAP ASE, the newly upgraded server retains the **abort tran on log full** setting.

Waking Suspended Processes

When **dump transaction** frees sufficient log space, suspended processes automatically awaken and complete.

- If **writetext** or **select into** has resulted in unlogged changes to the database since the last backup, run **dump tran with truncate_only**, which runs even in a situation with nonlogged-writes.
- If log space is so critical that **dump tran with truncate_only** fails, run **dump tran with no_log**. However, use **dump tran with no_log** for emergencies only, and run only as a last resort.

After the transaction dump completes, and transactions have successfully been freed from the log suspend state, the system administrator or database owner can dump the database.

If this does not free enough space to awaken the suspended processes, increase the size of the transaction log using the **log on** option of **alter database**.

As an alternative to killing the command, you can use **lct_admin("abort", *spid*)**, which might be preferable to killing the connection because you may want to maintain the connection. See the *Reference Manual: Building Blocks*.

If you have system administrator privileges, use **sp_who** to determine which processes are in a log suspend status, then use the **kill** command to waken the sleeping process.

Adding, Changing, and Deleting Thresholds

The database owner or system administrator can create additional thresholds to monitor free space on any segment in the database.

Additional thresholds are called *free-space thresholds*. Each database can have as many as 256 thresholds, including the last-chance threshold.

sp_addthreshold, **sp_modifythreshold**, and **sp_dropthreshold** allow you to create, change, and delete thresholds. All of these procedures require you to specify the name of the current database.

Displaying Information About Existing Thresholds

Use **sp_helpthreshold** to get information about all thresholds in a database. You can include a segment name with the stored procedure to get information about the thresholds on the specified segment.

The following example displays information about the thresholds on the database's default segment. Since "default" is a reserved word, you must enclose it in quotation marks. The output of **sp_helpthreshold** shows that there is one threshold on this segment set at 200 pages.

The 0 in the "last chance" column indicates that this is a free-space threshold instead of a last-chance threshold:

```
sp_helpthreshold "default"
segment name   free pages   last chance?   threshold procedure
------------   ----------   ------------   -------------------
default        200                     0   space_dataseg
```

Thresholds and System Tables

The system table `systhresholds` holds information about thresholds; **sp_helpthreshold** uses this table.

In addition to information about segment name, free page, last-chance status, and the name of the threshold procedure, the table also records the server user ID of the user who created the threshold and the roles the user had at the moment the threshold was created.

All system and user defined roles that are active when the threshold procedure is created are preserved in `systhresholds`. All system and user defined roles that the user had at time of creating the threshold and that have not been revoked are activated when the threshold fires.

SAP ASE gets information about how much free space is remaining in a segment—and whether to activate a threshold—from **curunreservedpgs**.

Creating Free-Space Thresholds

Use **sp_addthreshold** to create free-space thresholds.

See the *Reference Manual: Procedures*.

When the amount of free space on the segment falls below the threshold, an internal SAP ASE process executes the associated procedure. This process has the permissions of the user who created the threshold when he or she executed **sp_addthreshold**, less any permissions that have since been revoked.

Thresholds can execute a procedure in the same database, in another user database, in `sybsystemprocs`, or in `master`. They can also call a remote procedure on an Open Server. **sp_addthreshold** does not verify that the threshold procedure exists when you create the threshold.

Changing or Specifying a New Free-Space Threshold

Use **sp_modifythreshold** to associate a free-space threshold with a new threshold procedure, free-space value, segment, or change the name of the procedure associated with the last-chance threshold.

sp_modifythreshold drops the existing threshold and creates a new one in its place. See the *Reference Manual: Procedures*

For example, to execute a threshold procedure when free space on the segment falls below 175 pages rather than below 200 pages, enter:

```
sp_modifythreshold mydb, "default", 200, NULL, 175
```

In this example, NULL acts as a placeholder so that *new_free_space* falls in the correct place in the parameter list. The name of the threshold procedure does not change.

sp_modifythreshold requires that you specify the number of free pages associated with the last-chance threshold. Use **sp_helpthreshold** to determine this value

The person who modifies the threshold becomes the new threshold owner. When the amount of free space on the segment falls below the threshold, SAP ASE executes the threshold procedure with the owner's permissions at the time he or she executed **sp_modifythreshold**, less any permissions that have since been revoked.

This example displays information about the last-chance threshold, and then specifies a new procedure, **sp_new_thresh_proc**, to execute when the threshold is crossed:

```
sp_helpthreshold logsegment
```

```
segment name   free pages   last chance?   threshold procedure
------------   ----------   ------------   -------------------
logsegment     40                      1   sp_thresholdaction

(1 row affected, return status = 0)
```

```
sp_modifythreshold mydb, logsegment, 40, sp_new_thresh_proc
```

Dropping a Threshold

Use **sp_dropthreshold** to remove a free-space threshold from a segment.

The **sp_dropthreshold** *dbname* parameter must specify the name of the current database. You must specify both the segment name and the number of free pages, since there can be several thresholds on a particular segment.

For example:

```
sp_dropthreshold mydb, "default", 200
```

See the *Reference Manual: Procedures*.

Creating a Free-Space Threshold for the Log Segment

When the last-chance threshold is crossed, all transactions are aborted or suspended until sufficient log space is freed.

In a production environment, this can have a heavy impact on users. Adding a correctly placed free-space threshold on your log segment can minimize the chances of crossing the last-chance threshold.

The additional threshold should dump the transaction log often enough that the last-chance threshold is rarely crossed. It should not dump it so often that restoring the database requires the loading of too many tapes.

This section helps you determine the best place for a second log threshold. It starts by adding a threshold with a *free_space* value set at 45 percent of log size, and adjusts this threshold based on space usage at your site.

Use this procedure to add a log threshold with a *free_space* value set at 45 percent of log size.

1. Determine the log size in pages:

```
select sum(size)
from master..sysusages
where dbid = db_id("database_name")
and (segmap & 4) = 4
```

2. Use **sp_addthreshold** to add a new threshold with a *free_space* value set at 45 percent. For example, if the log's capacity is 2048 pages, add a threshold with a *free_space* value of 922 pages:

```
sp_addthreshold mydb, logsegment, 922, thresh_proc
```

3. Create a simple threshold procedure that dumps the transaction log to the appropriate devices.

See also

- *Creating Threshold Procedures* on page 432

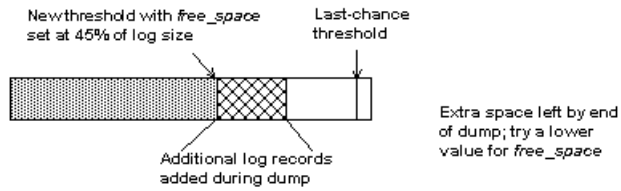
Usage Scenario: Testing and Adjusting the New Threshold

Use **dump transaction** to make sure your transaction log is less than 55 percent full.

1. Fill the transaction log by simulating routine user action. Use automated scripts that perform typical transactions at the projected rate.

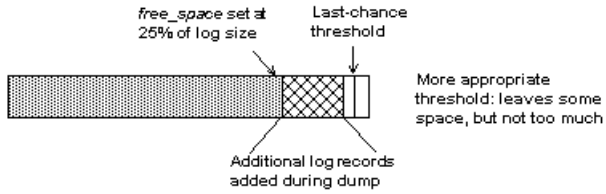
   When the 45 percent free-space threshold is crossed, your threshold procedure dumps the transaction log. Since this is not a last-chance threshold, transactions are not suspended or aborted; the log continues to grow during the dump.
2. While the dump is in progress, use **sp_helpsegment** to monitor space usage on the log segment. Record the maximum size of the transaction log just before the dump completes.
3. If considerable space was left in the log when the dump completed, you may not need to dump the transaction log so soon:

Figure 29: Transaction Log with Additional Threshold at 45 Percent



Try waiting until only 25 percent of log space remains:

Figure 30: Moving Threshold Leaves Less Free Space After Dump

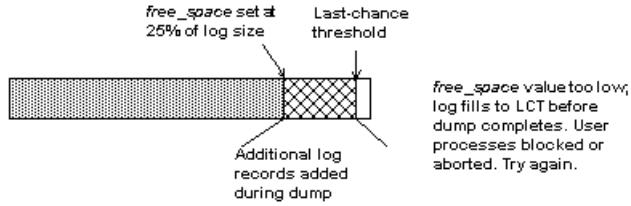


Use **sp_modifythreshold** to adjust the *free_space* value to 25 percent of the log size. For example:

```
sp_modifythreshold mydb, logsegment, 512,
    thresh_proc
```

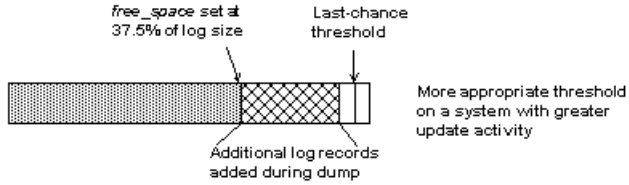
4. Dump the transaction log and test the new *free_space* value. If the last-chance threshold is crossed before the dump completes, you are not beginning the **dump transaction** soon enough:

Figure 31: Additional Log Threshold Does Not Begin Dump Early Enough



25 percent free space is not enough. Try initiating the dump transaction when the log has 37.5 percent free space:

Figure 32: Moving Threshold Leaves Enough Free Space to Complete Dump



Use **sp_modifythreshold** to change the *free_space* value to 37.5 percent of log capacity. For example:

```
sp_modifythreshold mydb, logsegment, 768,
    thresh_proc
```

Creating Additional Thresholds on Data and Log Segments

Create free-space thresholds on data segments as well as on log segments.

For example, you might create a free-space threshold on the default segment used to store tables and indexes. You would also create an associated stored procedure to print messages in your error log when space on the `default` segment falls below this threshold. If you monitor the error log for these messages, you can add space to the database device before your users encounter problems.

This creates a free-space threshold on the `default` segment of `mydb`. When the free space on this segment falls below 200 pages, SAP ASE executes a threshold procedure called **space_dataseg**:

```
sp_addthreshold mydb, "default", 200, space_dataseg
```

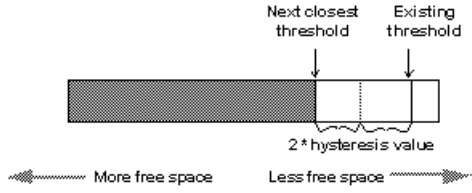
See also

- *Creating Threshold Procedures* on page 432

Determining Threshold Placement

Each new threshold must be at least twice the *@@thresh_hysteresis* value from the next closest threshold.

For example:



To see the hysteresis value for a database, issue:

```
select @@thresh_hysteresis
```

In this example, a segment has a threshold set at 100 pages, and the hysteresis value for the database is 64 pages. The next threshold must be at least 100 + (2 * 64), or 228 pages:

```
select @@thresh_hysteresis
-----------
         64
```

```
sp_addthreshold mydb, user_log_dev, 228,
sp_thresholdaction
```

Creating Threshold Procedures

SAP does not supply threshold procedures; create them yourself to ensure that they are tailored to your site's needs.

Suggested actions for a threshold procedure include writing to the server's error log and dumping the transaction log to increase the amount of log space. You can also execute remote procedure calls to an Open Server or to XP Server.

For example, if you include the following command in **sp_thresholdaction**, it executes a procedure named **mail_me** on an Open Server:

```
exec openserv...mail_me @dbname, @segment
```

See the *Transact-SQL Users Guide*.

This section provides some guidelines for writing threshold procedures, as well as two sample procedures.

See also

- *Creating a Free-Space Threshold for the Log Segment* on page 428
- *Creating Additional Thresholds on Data and Log Segments* on page 431

Parameters for Threshold Procedures

SAP ASE passes four parameters to a threshold procedure.

- *@dbname*, `varchar(30)`, which contains the database name
- *@segmentname*, `varchar(30)`, which contains the segment name
- *@space_left*, `int`, which contains the space-left value for the threshold
- *@status*, `int`, which has a value of 1 for last-chance thresholds and 0 for other thresholds

These parameters are passed by position rather than by name. Your procedure can use other names for these parameters, but must declare them in the order shown and with the datatypes shown.

Generating Error Log Messages

Include a **print** statement near the beginning of your procedure to record the database name, segment name, and threshold size in the error log.

If your procedure does not contain a **print** or **raiserror** statement, the error log will not contain any record of the threshold event.

The process that executes threshold procedures is an internal SAP ASE process. It does not have an associated user terminal or network connection. If you test your threshold procedures by executing them directly (that is, using **execute** *procedure_name*) during a terminal session, you see the output from the **print** and **raiserror** messages on your screen. When the same

procedures are executed by reaching a threshold, the messages, which include the data and time, go to the error log.

For example, if **sp_thresholdaction** includes this statement:

```
print "LOG DUMP: log for '%1!' dumped", @dbname
```

SAP ASE writes this message to the error log:

```
00: 92/09/04 15:44:23.04 server: background task message: LOG DUMP:
log for 'pubs2' dumped
```

sp_thresholdaction Procedures that Include a dump transaction

If your **sp_thresholdaction** procedure includes a **dump transaction** command, SAP ASE dumps the log to the devices named in the procedure.

dump transaction truncates the transaction log by removing all pages from the beginning of the log, up to the page just before the page that contains an uncommitted transaction record.

When there is enough log space, suspended transactions are awakened. If you abort transactions rather than suspending them, users must resubmit them.

If **sp_thresholdaction** failed because of nonlogged-writes status, you can issue **dump tran with no_log** as an alternative.

Generally, SAP recommends that you do not dump to one, especially to one that is located on the same machine or the same disk controller as the database disk. However, since threshold-initiated dumps can take place at any time, you may want to dump to disk and then copy the resulting files to offline media. (You must copy the files back to the disk to reload them.)

Your choice depends on:

- Whether you have a dedicated dump device online, loaded and ready to receive dumped data
- Whether you have operators available to mount tape volumes during the times your database is available
- The size of your transaction log
- Your transaction rate
- Your regular schedule for dumping databases and transaction logs
- Available disk space
- Other site-specific dump resources and constraints

A Simple Threshold Procedure

Youc an create a simple procedure that dumps the transaction log and prints a message to the error log.

Because this procedure uses a variable (*@dbname*) for the database name, it can be used for all databases in SAP ASE:

```
create procedure sp_thresholdaction
    @dbname varchar(30),
```

```
    @segmentname varchar(30),
    @free_space int,
    @status int
as
dump transaction @dbname
    to tapedump1
print "LOG DUMP: '%1!' for '%2!' dumped",
        @segmentname, @dbname
```

A More Complex Procedure

You can create a completx threshold procedure that performs different actions, depending on the value of the parameters passed to it.

Its conditional logic allows it to be used with both log and data segments.

The procedure:

- Prints a "LOG FULL" message if the procedure was called as the result of reaching the log's last-chance threshold. The status bit is 1 for the last-chance threshold, and 0 for all other thresholds. The test **if (@status&1) = 1** returns a value of true only for the last-chance threshold.
- Verifies that the segment name provided is the log segment. The segment ID for the log segment is always 2, even if the name has been changed.
- Prints "before" and "after" size information on the transaction log. If a log does not shrink significantly, it may indicate that a long-running transaction is causing the log to fill.
- Prints the time the transaction log dump started and stopped, helping gather data about dump durations.
- Prints a message in the error log if the threshold is not on the log segment. The message gives the database name, the segment name, and the threshold size, letting you know that the data segment of a database is filling up.

```
create procedure sp_thresholdaction
     @dbname           varchar(30),
     @segmentname      varchar(30),
     @space_left       int,
     @status           int
as
declare @devname varchar(100),
           @before_size int,
           @after_size int,
           @before_time datetime,
           @after_time datetime,
           @error int

/*
** if this is a last-chance threshold, print a LOG FULL msg
** @status is 1 for last-chance thresholds,0 for all others
*/
if (@status&1) = 1
begin
      print "LOG FULL: database '%1!'", @dbname
end
```

```
/*
** if the segment is the logsegment, dump the log
** log segment is always "2" in syssegments
*/
if @segmentname = (select name from syssegments
               where segment = 2)
begin
     /* get the time and log size
     ** just before the dump starts
     */
     select  @before_time = getdate(),
     @before_size = reserved_pages(db_id(), object_id("syslogs"))
      print "LOG DUMP: database '%1!', threshold '%2!'",
        @dbname, @space_left
      select @devname = "/backup/" + @dbname + "_" +
        convert(char(8), getdate(),4) + "_" +
        convert(char(8), getdate(), 8)
      dump transaction @dbname to @devname
      /* error checking */
      select @error = @@error
      if @error != 0
      begin
           print "LOG DUMP ERROR: %1!", @error
      end
/* get size of log and time after dump */
    select  @after_time = getdate(),
      @after_size = reserved_pages(db_id(), object_id("syslogs"))
      /* print messages to error log */
      print "LOG DUMPED TO: device '%1!", @devname
      print "LOG DUMP PAGES: Before: '%1!', After '%2!'",
           @before_size, @after_size
      print "LOG DUMP TIME: %1!, %2!", @before_time, @after_time
end
        /* end of 'if segment = 2' section */
      else
       /* this is a data segment, print a message */
      begin
      print "THRESHOLD WARNING: database '%1!', segment '%2!' at '
%3!' pages", @dbname, @segmentname, @space_left
end
```

Placement for Threshold Procedures

Although you can create a separate procedure to dump the transaction log for each threshold, it is easier to create a single threshold procedure that is executed by all log segment thresholds.

When the amount of free space on a segment falls below a threshold, SAP ASE reads the `systhresholds` table in the affected database for the name of the associated stored procedure, which can be any of:

- A remote procedure call to an Open Server
- A procedure name qualified by a database name (for example, `sybsystemprocs.dbo.sp_thresholdaction`)

- An unqualified procedure name

If the procedure name does not include a database qualifier, SAP ASE looks in the database where the space shortage occurred. If it cannot find the procedure there, and if the procedure name begins with the characters “sp_”, SAP ASE looks for the procedure in the `sybsystemprocs` database and then in `master` database.

If SAP ASE cannot find the threshold procedure, or cannot execute it, it prints a message in the error log.

Disabling Free-Space Accounting for Data Segments

Use the **no free space acctg** option to **sp_dboption** to disable free-space accounting on data segments. You cannot disable free-space accounting on the log segment.

Setting **no free space acctg** to **true** helps SAP ASE start faster after an abnormal shutdown or a **shutdown with nowait**. However, you should use **no free space acctg** only if you do not have thresholds on your data segments or mixed log and data.

When you disable free-space accounting, only the thresholds on your log segment monitor space usage; threshold procedures on your data segments do not execute when thresholds are crossed. Disabling free-space accounting speeds recovery time because free-space counts are not recomputed during recovery for any segment except the log segment.

After you disable data segment free-space accounting, the counts may be inaccurate, even after you set **no free space acctg** to false. Issuing **shutdown with nowait** and restarting SAP ASE forces SAP ASE to recalculate the free-space accounting, and may increase recovery time.

This example turns off free-space accounting for the `production` database:

```
sp_dboption production,
     "no free space acctg", true
```

CHAPTER 20 Transaction Log Space Management

Analyze and free transaction log space.

SAP ASE provides a single transaction log segment per database. Space can be added to a log segment or removed from a log segment. However, at any given point, there is limited space in a log segment.

Whenever the database client applications perform any data manipulation language (DML) operations on the data, SAP ASE produces log records that consume space in the transaction log. Typically there are several clients performing DMLs concurrently on the database and log records are appended to the log whenever user log caches (ULCs) for individual clients are full or in some other conditions such as when a data page is changed by multiple open transactions. Log records of several transactions are therefore typically interleaved in the transaction log space.

Removing transactions from the transaction log to free the log space can be done using **dump transaction**. However, there different scenarios that can cause the transaction log can grow in such a way that the **dump transaction** command is not able to free space. In these situations, the log consumes space to such an extent that it affects the continuous availability of the database system for any write operations. The scenarios include:

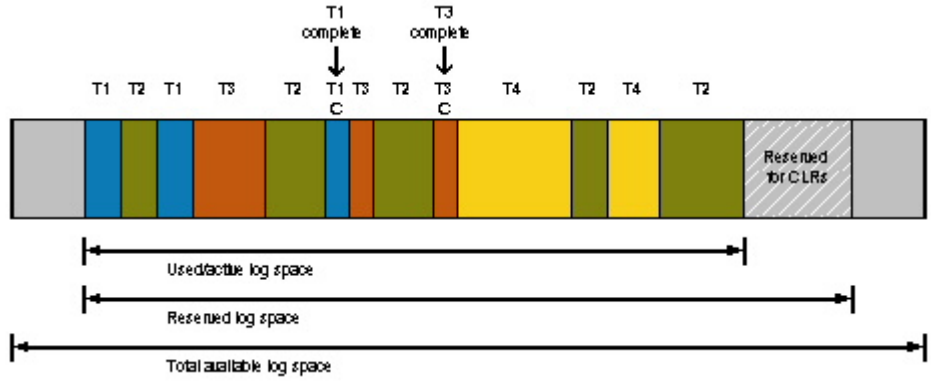
- Transactions are entered into the server but not committed. In this situation the log cannot be truncated because there is an active transaction.
- Replication Server is slow in reading the log which prevents truncating the log.
- A dump transaction has not been performed for a long period of time. Periodically dumping transactions can keep the size of the reserved space in the log limited and ensure that there is free space available in the log which allows the space freed after dump transaction to be reused for further logging activity.

You can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free log space. The **sp_thresholdaction** procedure can be used to free log space in the transaction log if the available free space falls below a preconfigured threshold. The recommended action is to define a trigger that will execute **dump transaction** once the log fall below the threshold. However, the **dump transaction** command cannot truncate the portion of the log beginning from of the oldest incomplete or active transaction in the log, since this portion is needed for recovery of the database. In this case, the oldest transactions can be aborted, depending on circumstances.

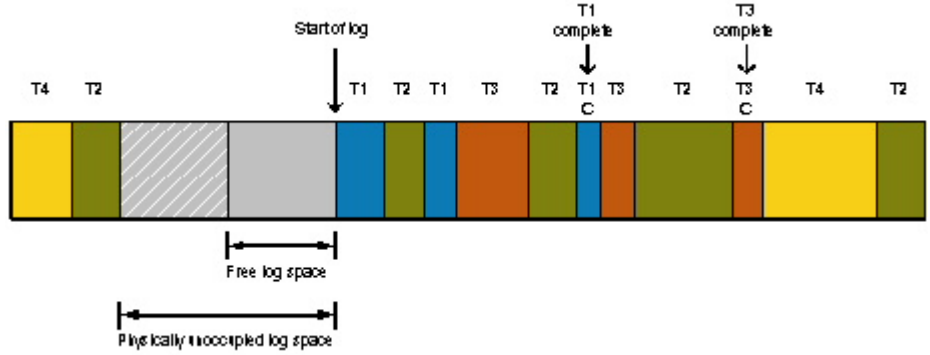
Transaction Log Space

The transaction log contains sufficient information about each transaction to ensure that it can be recovered. This transaction information can be interleaved within the log.

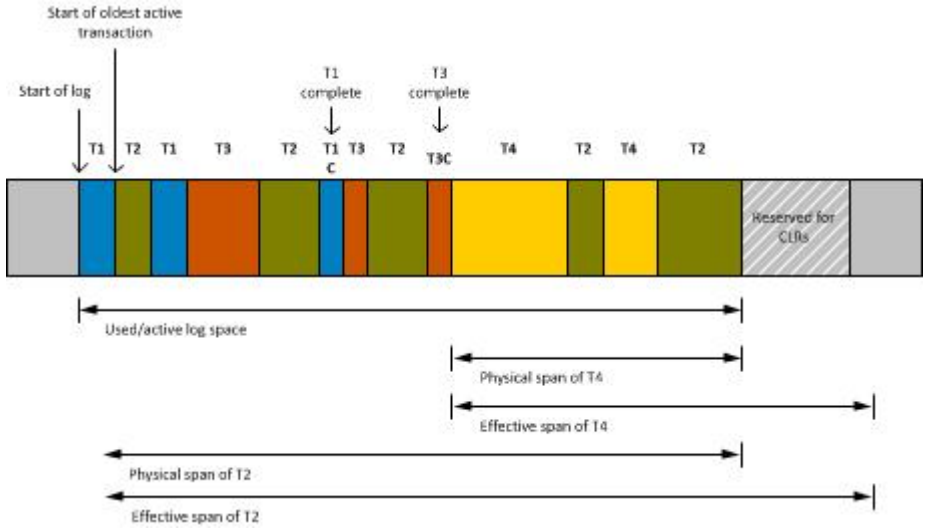
This transaction log shows the space allocation of the transaction log.



If a total allocated log segment is considered a contiguous logical space, then additional log records are appended to the end of the log. When there is no more space available at the end of the log, portions of the log that have been freed in the beginning of allocated space will be used to add records. This use of the freed portion of log can introduce a circular or wrapped around form within the allocated space.



The span of a completed transaction in the log is the portion of a log between begin log record of the transaction in the log and end log record of the transaction. The actual logging done by a transaction could be very minimal, however due to intermediate interleaved log records by other transaction, this span in terms of transaction log space could be considerable.



As transaction log space allocation is done in terms of pages, the span of an active transaction or incomplete transaction is also defined in terms of the number of log pages. There can be many active transactions at any point in the log. The portion of the transaction log occupied by span of an active transaction cannot be truncated with **dump transaction**. The total portion of the log which cannot be truncated with **dump transaction** is equal to span of the oldest active transaction

Automating Transaction Log Management

You can use **sp_thresholdaction** to identify the oldest active transaction and use **sp_xact_loginfo** to monitor the longest running transaction per database or abort the oldest active transaction based on conditional criteria.

Rescue Scenario Use Case

sp_thresholdaction can be used to identify the oldest active transaction and decide on action based on the information returned.

You can truncate the log to free up the space or abort the oldest active transaction or both based on the defined criteria. This use case assumes that the oldest active transaction span needs to be watched or limited only when free space in log segment falls beyond threshold.

```
create procedure sp_logthresholdaction
        @dbname         varchar(30),
        @segmentname    varchar(30),
        @space_left     int,
        @status         int
as
declare
```

```
        @oats_span_pct          int,
        @dbid                   int,
        @at_startpage           bigint,
        @firstlog_page          bigint,
        @canfree_withoutabort   int,
        @stp_blocking           int,
        @stp_pagebig            int,
        @dump_in_progress       int,
                @oat_spid               int,
                @oat_starttime          datetime,
                @activexact             int,
                @free_with_dumptran_pct int,
                @errorcode              int,
                @xactspan_limit         int,
                @space_left_pct         int,
                @dumptrancmd            varchar(256),
                @dumpdevice             varchar(128),
                @killcmd                varchar(128)

        select @dbid = db_id(@dbname)
        select @dumpdevice = "/ferrum_dev1/db1.tran.dmp1"

        select @free_with_dumptran_pct = 5
/*
** attempt dump tran only if it can free at
** least 5 percent of log space
*/
        select @xactspan_limit = 20
/*
** optionally also kill oldest active transaction even after
** dump tran if its exceeds 20 percent in the log
*/

    select @space_left_pct = logspace_pagestopct(@dbid, @space_left, 0)
    print "Space left in log segment " + @space_left_pct + " percent."

    select @dump_in_progress = 1
    while (@dump_in_progress> 0)
        begin    -- {
            exec sp_xact_loginfo@dbid,
            "oldestactive",
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output
```

```
if (@dump_in_progress> 0)
begin
    sleep 30
    continue
end
select @killcmd = "kill " + @xact_spid + " with status_only"

if (@canfree_withoutabort>@free_with_dumptran)
begin
     select @dumptrancmd = "dump tran " + @dbname + " on " + @dumpdevice
     exec(@dumptrancmd)
/*
** Optionally, also abort oldest active transaction.
*/
        if ((@stp_blocking = 0) and
        (@oats_span_pct> @xactspan_limit))
        then
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
            exec(@killcmd)
        end
else
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
    exec(@killcmd)
    end
    end -- }
```

Monitoring Use Case

sp_xact_loginfo can be used for periodically monitoring the longest running transaction per database.

For example, a stored procedure can be formed around **sp_xact_loginfo** in which it populates the tables with the oldest active transaction information and populates a user defined table. The execution of this stored procedure can be periodic, at desired frequency through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table
** for monitoring oldest active transactions with following
** table definition:
**
**    create table oldest_active_xact(
**           dbid                    int,
**           oldestactivexact_span   int,
**           startxactpagenum        int,
**           spid                    int,
**           xactstarttime           varchar(27),
**           startlogpagenum         int,
**           stppage                 bigint,
```

```
**           sec_truncpoint_block     int,
**           can_free_wo_kill         int,
**           dump_in_progress         int,
**           nolog                    int,
**           username                 varchar(30) null)
*/
create procedure sp_record_oldestactivexact
@dbname    varchar(30)
as
declare     @dbid int,
            @oats_span_pct int,
            @oat_startpage bigint,
            @firstlog_page bigint,
            @canfree_withoutabort int,
            @stp_blocking int,
            @stp_page bigint,
            @dump_in_progress int,
            @oat_spid int,
            @oat_starttime varchar(27),
            @activexact int,
            @free_with_dumptran_pct int,
            @errorcode int,
            @username varchar(30)

select @dbid = db_id(@dbname)

exec sp_xact_loginfo @dbid,
            'oldestactive',
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output

if (@activexact = 1)
    begin
        print "activexact is true"
        select @username = suser_name(sysproc.uid)
                from master..systransactions systran,
                master..sysprocesses sysproc
                      where systran.spid = @oat_spid
                and systran.spid = sysproc.spid
        insert into oldest_active_xact
            values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
```

```
                @oat_spid,
                @oat_starttime,
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,@username)
    end
else
    begin
        print "activexact is false"
        insert into oldest_active_xact values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
                @oat_spid,getdate(),
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,NULL)
end
```

Monitoring and Control Use Case

In addition to monitoring, the action of aborting the oldest active transaction based on conditional criteria can also be implemented in **sp_xact_loginfo** which is run periodically through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table for
monitoring
** oldest active transactions with following table definition:
**
**    create table oldest_active_xact(datetimetime_of_recording,
**           dbid                      int,
**           oldestactivexact_span     int,
**           spid                      int,
**           username                  varchar(30),
**           startxactpagenum          int,
**           startlogpagenum           int,
**           xactstarttime             datetime,
**           can_free_wo_kill          int,
**           sec_truncpoint_block      int,
**           nolog                     int,
**           action_taken              varchar(30))
**           lock datarows
*/
create procedure sp_control_oldestactivexact
          @dbname    varchar(30)
as
declare    @oats_span_pct           int,
           @dbid                    int,
```

```
            @at_startpage            bigint,
            @firstlog_page           bigint,
            @canfree_withoutabort    int,
            @stp_blocking            int,
            @stp_pagebig             int,
            @dump_in_progress        int,
            @oat_spid                int,
            @oat_starttime           datetime,
            @activexact              int,
            @free_with_dumptran_pct  int,
            @errorcode               int,
            @username                varchar(30),
            @action_taken            varchar(30),
            @xact_maxspan_pct        int,
            @killcmd                 varchar(128)

    select @dbid = db_id(@dbname)
    select @xact_maxspan_pct = 20
    select @action_taken = "none"

exec sp_xact_loginfo @dbid,
        "oldestactive",
        NULL,
        0,
        0,
@span_pct = @oats_span_pct output,
@startpage = @oat_startpage output,
@xact_spid = @oat_spid output,
@starttime = @oat_starttime output,
@firstlog_page = @firstlog_page output,
@stp_page = @stp_page output,
@stp_blocking = @stp_blocking output,
@canfree_without_abort_pct = @free_with_dumptran_pct output,
@dump_in_progress = @dump_in_progress output,
@activexact = @activexact output,
@errorcode = @errorcode output

    select @killcmd = "kill " + @oldesactive_spid + " with status_only"

    if (@nolog == 0)
    then
select @username = suser_name(systran.suid)
from master..systransactionssystran where systran.spid
=@oldestactive_spid
if (@oats_span_pct> @xact_maxspan_pct)
begin
    exec(@killcmd)
    select @action_taken = "transaction abort"
end
        insert into oldest_active_xact values(getdate(), @dbid,
@oats_span_pct, @oat_spid, @username, @oat_page, @firstlog_page,
@free_with_dumptran_pct, @stp_blocking, @activexact, @action_taken)
    else
        /*
        ** Just to cover possibility of no active transactions which have
        ** generated any log.
```

```
        */
        insert into oldest_active_xact values(getdate(), @dbid,
                    0, 0, NULL, 0, 0, 0, 0, 1, @action_taken)
    end
```

Analyzing and Managing Transaction Log Space

Use the **loginfo** function to view and free transaction log space.

The system administrator can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free space.

This example uses **loginfo** to show the transaction log at a particular point in time:

```
======================================
select loginfo(dbid, 'database_has_active_transaction') as has_act_tran,
       loginfo(dbid, 'oldest_active_transaction_pct') as Act_log_portion_pct,
       loginfo(dbid, 'oldest_active_transaction_spid') as OA_tran_spid,
       loginfo(dbid, 'can_free_using_dump_tran') as dump_tran_free_pct,
       loginfo(dbid, 'is_stp_blocking_dump') as is_stp_blocking,
       loginfo(dbid, 'stp_span_pct') as stp_span_pct
```

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ----------- ------------------- ------------------ --------------- ------------ ---------------
           1          19                  38                  7               0           45              52

(return status = 0)
```

This shows:

- `has_act_tran` = 1, indicates that the database currently has one active transaction.
- `OA_tran_spid` = 19, indicates that the system process ID of the oldest active transaction in the database is 19.
- `Act_log_portion_pct` = 38, indicates that 38 percent of the log space is occupied by the oldest active transaction.
- `dump_tran_free_pct` = 7, indicates that 7 percent of the transaction log that can freed using **dump transaction**.
- `is_stp_blocking` = 0, indicates that there is no secondary truncation point preventing the use of **dump transaction** to free space.
- `stp_span_pct = 45`, indicates that there is a secondary truncation point spanning 45 percent of the transaction log.
- `log_occupied_pct` = 52, indicates that 52 percent of the total log space is currently occupied.

The available actions are:

1. The first step can be to use **dump transaction** to free the transaction log of the 7 percent shown by `dump_tran_free_pct = 7`. After freeing the space using **dump transaction**, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ----------- ------------------- ------------------ --------------- ------------ ---------------
```

```
      1          19              38               0               1             45           45

(return status = 0)
```

2. At this stage, `Act_log_portion_pct = 38`, indicates that 38 percent of the log space is occupied by the transaction with the system process ID of 19. You can wait for system process 19 to complete, or abort the transaction.

   If you decide to abort the transaction using the **kill** command (with or without status only option) as a measure to rescue the log, re-issuing the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------ ------------------ ---------------- ------------ ---------------
       0            0               0                 45                0             0            45

(return status = 0)
```

3. The query shows that there are no active transaction in the system. You can free all 45 percent of the occupied log space using the **dump transaction** command. After **dump transaction** is executed, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------ ------------------ ---------------- ------------ ---------------
       0            0               0                 0                 0             0            0

(return status = 0)
```

Viewing the Span of a Transaction

The system administrator can view the span of a transaction started by a specific process.

In this example, the transaction is identified by system process ID 15 and the database ID is 4:

```
select loginfo(4, 'xactspanbyspid', 15)
    as xact_span_spid15_dbid4
```

```
xact_span_spid15_dbid4
 ----------------------
                     10
```

This indicates that system process 15 is an active transaction and the log transaction span is 10 percent.

Viewing the Oldest Active Transactions

The system administrator can view the processes that are using the most log space.

This example shows the top three oldest active transactions having longest spans in the transaction log:

```
select top 3 convert(numeric(3,0),
      loginfo(db_id(), 'xactspanbyspid', t.spid)) as XACTSPAN,
          convert(char(4), t.spid) as SPID,
          convert(char(20), t.starttime) as STARTTIME,
          convert(char(4), p.suid) as SUID,
          convert(char(15), p.program_name) as PROGNAME,
          convert(char(15), p.cmd) as COMMAND,
          convert(char(16), p.hostname) as HOSTNAME,
          convert(char(16), p.hostprocess) as HOSTPROCESS
```

```
from master..systransactions t, master..sysprocesses p
where t.spid = p.spid
order by XACTSPAN desc
```

```
XACTSPAN SPID     STARTTIME     SUID PROGRAM COMMMAND  HOSTNAME
HOSTPROCESS
-------- ---- ----------------- ---- ------- -------- ----------
-----------
  38      19  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore4
26141
  20      20  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore2
23467
  10      21  Aug 5 2013 12:21AM  1    ISQL   WAITFOR linuxstore6
4971
(return status =0)
```

Truncating a Log that Is Not on A Separate Segment

If a database does not have a log segment on a device that is separate from data segments, you cannot use **dump transaction** to copy the log and then truncate it.

1. Use the special **with truncate_only** option of **dump transaction** to truncate the log so that it does not run out of space. Because it does not copy any data, with **truncate_only** requires only the name of the database.
2. Use **dump database** to copy the entire database, including the log.

This example dumps the database `mydb`, which does not have a log segment on a separate device from data segments, and then truncates the log:

```
dump database mydb to mydevice
dump transaction mydb with truncate_only
```

Truncating the Log in Early Development Environments

In early development environments, the transaction log can be quickly filled by creating, dropping, and re-creating stored procedures and triggers, and checking integrity constraints.

Recovery of data may be less important than ensuring that there is adequate space on database devices.

with truncate_only lets you truncate the transaction log without making a backup copy:

```
dump transaction database_name with truncate_only
```

After you run **dump transaction with truncate_only,** you must dump the database before you can run a routine log dump.

Truncating a Log that Has No Free Space

When the transaction log is very full, you may not be able to use your usual method to dump it.

If you used **dump transaction** or **dump transaction with truncate_only**, and the command failed because of insufficient log space, use the **with no_log** option of **dump transaction**:

```
dump transaction database_name with no_log
```

This option truncates the log without logging the dump transaction event. Because it does not copy any data, it requires only the name of the database.

Warning! Use **dump transaction with no_log** as a last resort, and use it only once after **dump transaction with truncate_only** fails. If you continue to load data after entering **dump transaction with no_log**, you may fill the log completely, causing any further **dump transaction** commands to fail. Use **alter database** to allocate additional space to the database.

All occurrences of **dump tran with no_log** are reported in the SAP ASE error log. Messages indicating success or failure are also sent to the error log. **no_log** is the only dump option that generates error log messages.

Dangers of Using with truncate_only and with no_log

The **with truncate_only** and **with no_log** parameters allow you to truncate a log that has become disastrously short of free space.

Neither option provides a means to recover transactions that have committed since the last routine dump.

Warning! Run **dump database** at the earliest opportunity to ensure that your data can be recovered.

This example truncates the transaction log for `mydb` and then dumps the database:

```
dump transaction mydb
    with no_log
dump database mydb to ...
```

Provide Sufficient Log Space

Every use of **dump transaction...with no_log** is considered an error and is recorded in the server's error log.

If you have created your databases with log segments on a separate device from data segments, written a last-chance threshold procedure that dumps your transaction log often enough, and allocated enough space to your log and database, you should not have to use this option.

However, some situations can still cause the transaction log to become too full, even with frequent log dumps. The **dump transaction** command truncates the log by removing all pages

from the beginning of the log, up to the page preceding the page that contains an uncommitted transaction record (known as the oldest active transaction). The longer this active transaction remains uncommitted, the less space is available in the transaction log, since **dump transaction** cannot truncate additional pages.

This can happen when applications with very long transactions modify tables in a database with a small transaction log, which indicates you should increase the size of the log. It also occurs when transactions remain uncommitted for long periods of time, such as when an implicit **begin transaction** uses the chained transaction mode, or when a user forgets to complete the transaction. You can determine the oldest active transaction in each database by querying the `syslogshold` system table.

Querying the `syslogshold` Table

The `syslogshold` table is located in the `master` database.

Each row in the table represents either:

- The oldest active transaction in a database, or
- The Replication Server truncation point for the database's log.

A database may have no rows in `syslogshold`, a row representing one of the above, or two rows representing both of the above. For information about how a Replication Sever truncation point affects the truncation of the database's transaction log, see the Replication Server documentation.

Querying `syslogshold` provides a snapshot of the current situation in each database. Since most transactions last for only a short time, the query's results may be inconsistent. For example, the oldest active transaction described in the first row of `syslogshold` may finish before SAP ASE completes the query of `syslogshold`. However, when several queries of `syslogshold` over time query the same row for a database, that transaction may prevent a **dump transaction** from truncating any log space.

When the transaction log reaches the last-chance threshold, and **dump transaction** cannot free space in the log, you can query `syslogshold` and `sysindexes` to identify the transaction holding up the truncation. For example:

```
select H.spid, H.name
from master..syslogshold H, threshdb..sysindexes I
where H.dbid = db_id("threshdb")
and I.id = 8
and H.page = I.first
```

```
spid    name
------  ------------------------------------
     8  $user_transaction

(1 row affected)
```

This query uses the object ID associated with `syslogs` (8) in the `threshdb` database to match the first page of its transaction log with the first page of the oldest active transaction in `syslogshold`.

You can also query `syslogshold` and `sysprocesses` in the `master` database to identify the specific host and application owning the oldest active transactions.

For example:

```
select P.hostname, P.hostprocess, P.program_name,
   H.name, H.starttime
from sysprocesses P, syslogshold H
where P.spid = H.spid
and H.spid != 0
```

```
hostname hostprocess program_name name                starttime
-------- ----------- ------------ ------------------ ---------------
---
eagle         15826 isql         $user_transaction  Sep  6 1997 4:29PM
hawk          15859 isql         $user_transaction  Sep  6 1997 5:00PM
condor        15866 isql         $user_transaction  Sep  6 1997 5:08PM

(3 rows affected)
```

Using the above information, you can notify or kill the user process that owns the oldest active transaction and proceed with the **dump transaction**. You can also include the above types of queries in the threshold procedures for the database as an automatic alert mechanism. For example, you may decide that the transaction log should never reach its last-chance threshold. If it does, your last-chance threshold procedure (**sp_thresholdaction)** alerts you with information about the oldest active transaction preventing the transaction dump.

Note: The initial log records for a transaction may reside in a user log cache, which is not visible in `syslogshold` until the records are flushed to the log (for example, after a checkpoint).

See also

- *Chapter 19, Managing Free Space with Thresholds* on page 419