New Features Guide

Adaptive Server Enterprise

15.7 ESD #2

DOCUMENT ID: DC00641-01-1572-01

LAST REVISED: June 2012

Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

CHAPTER 1 Deferred Table Creation

The with deferred_allocation parameter for the create table command lets you defer page allocation for a table. Deferred tables help applications that create numerous tables, but use only a small number of them. Tables are called “deferred” until Adaptive Server ® allocates their pages.

System tables include entries for deferred tables. These entries allow you to create objects associated with deferred tables such as views, procedures, triggers, and so on.

Adaptive Server performs page allocation for deferred tables when it inserts the first row (called table materialization). Access to the table before the first insert, such as selects, deletes or updates, functions that report space usage, or enforce referential integrity constraints during DML on other tables, behave as if the table is empty. That is, a select against a deferred table produces an empty result set. Although you can create indexes on deferred tables, the page allocation for these indexes is deferred until Adaptive Server materializes the table.

Setting deferred table creation at the database level

Use the ‘deferred table allocation' database option to configure the database to defer page allocation for all subsequently created user tables:

```
sp_dboption database_name, "deferred table allocation", true
```

You cannot enable deferred table allocation for any system databases (such as master, sybsystemprocs, sybsystemdb) or temporary databases.

Creating deferred tables

Use the create table . . . with deferred_allocation parameter to create deferred tables:

create table *table_name* . . . with deferred_allocation

For example, to create a table named im_not_here_yet, enter:

```
create table im_not_here_yet (
col_1 int,
col_2 varchar(20)
)
with deferred_allocation
```

sp_dboption 'deferred table allocation' need not be enabled to create deferred tables.

Use create table . . . with immediate_allocation to create tables that are not deferred when sp_dboption 'deferred table allocation' is enabled. The syntax is:

create table *table_name* . . . with immediate_allocation

Explicitly materializing deferred tables

Use alter table . . . immediate_allocation to explicitly materialize a deferred table. The syntax is:

alter table *table_name* immediate_allocation

Once you materialize the table, Adaptive Server allocates pages for all data and index partitions.

For example, to materialize the table im_not_here_yet, enter:

```
alter table im_not_here_yet immediate allocation
```

Identifying deferred tables

sp_help includes information about deferred tables in the object_status column. This example shows a partial sp_help output for the im_not_here_yet deferred table:

```
sp_help im_not_here_yet
Name            Owner  Object_type  Object_status        Create_date
---------------  -----  -----------  -------------------  -------------------
im_not_here_yet    dbo   user table  deferred allocation    Apr 9 2012 2:09PM
```

sysobjects includes the 0x80 status bit in the sysstat3 column to indicate that a table is deferred.

Rolling back for deferred tables

If the materialization of a deferred table is part of a transaction that gets rolled back, Adaptive Server does not roll back the page allocation it performed for the deferred table.

For example:

```
create table im_not_here_yet with deferred_allocation
go
begin tran t1
go
insert into deferred table ...
go
rollback tran t1
```

insert materializes the im_not_here_yet, table, then inserts a value. Although the rollback tran removes the value from the table, the page allocation is not rolled back, so the table remains materialized and is no longer a deferred table.

How commands behave in deferred tables

Most commands work similarly on deferred tables and empty tables:

Command	Action on deferred table
insert	Materializes the table; execute insert
select	0 rows selected
update	0 rows affected
delete	0 rows affected
alter table	Materialize the table; execute alter table
drop table	Drop table
create view, trigger or, procedure	Creates view, trigger, or procedure
create index	Creates indexes without page allocations
drop index	Drops index
reorg subcommands	None

Command	Action on deferred table
update statistics	None
truncate table	None
dbcc checktable	None
dbcc checkcatalog	Skips indexes on deferred tables

CHAPTER 2 Concurrency Enhancements

The reorg rebuild command in Adaptive Server version 15.7 ESD # 2 and later includes an online parameter that lets you reorganize data and perform maintenance without blocking the data from users.

reorg rebuild

reorg rebuild ... with online allows you to reorganize your data without taking the data offline. The syntax is:

```
reorg rebuild table_name
    [with online]
```

For example, to rebuild the indexes on the titles table and keep the data online, enter:

```
reorg rebuild titiles 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

Sybase® recommends that you run reorg rebuild ...online when the table's transaction load is relatively low.

Recovering

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.

Restrictions

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 ... with online is running. Adaptive Server does not allow inserts that lead to page splits on tables with allpages 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.

CHAPTER 3 Merging and Splitting Partitions

Over time, a partition’s data distribution may become skewed, or the manner in which the data was originally partitioned may not suit the current business requirements. Use alter table to merge, split, or move partitions to redistribute the data and revive performance benefits using partitions.

For example, a company may split partitions to better access its data according to four regions —North, South, East and West. The split partitions allow customer representatives fast and efficient access to their regions’ customers, independent of other regions. If sales increase in the Southern region and the customer base has expanded significantly, frequent queries involving partition scans and maintenance operations may cause the South partition to be slow and inefficient, losing out on the benefits of partitioning the customer data. In this situation, splitting the data in the South partition into two partitions, South-East and South-West, may revive performance without affecting the data in other partitions.

A company may merge partitions for better performance because their sales data is partitioned into the four yearly quarters—partitions Q1, Q2, Q3, and Q4. At the end of the year, the company merges the data for the year and archives it. Merging the partitions is efficient because the sales data for a past year is accessed infrequently, and the older data is most likely to be read but not updated.

Partition schemes available for splitting or merging

alter table allows you to split, merge, or move these partitioning schemes:

- Range partitioning
- List partitioning

Because the location for the data in round-robin partitioned tables is distributed randomly among the data partitions, there is no need to split or merge round-robin partitions.

For hash-partitioned tables, use the repartition utility to redistribute the data.

Splitting partitions

Use the alter table ... split partition parameter to redistribute data to two or more partitions. The syntax is:

alter table *table_name*
split partition *partition_name*
into *partition_condition_clause*

where:

- *partition_name* – the partition you are splitting.
- *partition_condition_clause* – conditions that specify how to split the source partition data. Typically, conditions consist of a numerical range or a data range. The partition conditions should cover all, and only, the data in the source partition.

  partition_condition_clause may be on the same segment as the source partition, or on a new segment. If you do not specify destination partition segments, Adaptive Server creates the new partitions on the segment on which the source partition resides.

See *Reference Manual: Commands*.

You must enable select into/bulkcopy to issue alter table ... split partition. By default, alter table ... split partition rebuilds the section of the local or global index on the partitioned table affected by the split operation.

Except for the step that rebuilds the index, alter table ... split partition is not a logged operation. Sybase recommends that you perform a database dump after running the alter table ... split partition command.

This example creates the orders table and then splits its partitions to redistribute the data:

```
create table orders (orderid int, amount float,
orderdate datetime)
partition by range (amount)
( P1 values <= (10000) on seg1,
  P2 values <= (50000) on seg2,
  P3 values <= (100000) on seg3,
  P4 values <= (MAX) on seg4)

create clustered index ind_orderid
on orders(orderid) local index (i1 on seg1, i2 on seg2,
i3 on seg3, i4 on seg4)

alter table orders
```

```
split partition P2
into
( P5 values <= (25000) on seg2,
  P6 values <= (50000) on seg3)

alter table orders
split partition P3
into
( P7 values <= (50000) on seg2,
  P8 values <= (100000) on seg3)

alter table orders
split partition P4
into
( P9 values <= (200000),
  P10 values <= (MAX))
```

Merging partitions

Use alter table ... merge partition to combine the data from two or more merge-compatible (that is, available for the merge) partitions into a single partition. Whether partitions are merge compatible depends on how they are partitioned:

- For list-partitioned tables, any two partitions are merge-compatible
- For range-partitioned tables, partitions must be adjacent to be merge-compatible

The syntax is:

alter table *table_name*
merge partition {*partition_name* [{, *partition_name*}…]}
into *destination_partition_name* [on *segment_name*]

where:

- *partition_name* – the source partitions you are merging. All source partitions must be on the same segment.
- *destination_partition_name* – a new or existing partition. If *destination_partition_name* is an existing partition, it cannot be any of the source partitions you are merging.

The partition condition for the merged destination partition is derived from the partition conditions of all the source data partitions being merged, so the destination partition includes all the data residing in the source data partitions being merged. For example, for a list-partitioned table, the new partition condition for the merged partition is the union of all the values that form the source data partition conditions.

See *Reference Manual: Commands*.

You must enable select into/bulkcopy to issue alter table ... merge partition.

alter table ... merge partition is fully logged. Use the transaction dump to recover from a server failure.

This example creates the sales table and then merges its partitions to consolidate the data:

```
create table sales(salesmanid int, salesdate datetime,
salesregion varchar(10))
partition by range(salesdate)
( Q1 values <= ('31 Mar 2007'),
  Q2 values <= ('30 Jun 2007'),
  Q3 values <= ('30 Sep 2007'),
  Q4 values <= ('31 Dec 2007'))
create index ind_region on sales(salesregion)

alter table sales
merge partition Q3
into Q4

alter table sales
merge partition Q1, Q2, Q3, Q4
into Y2007
```

Moving partitions

Use alter table ... move partition to move a partition (and its index) to a specified segment. The syntax is:

alter table *table_name*
move partition *partition_name*
to *destination_segment_name*

where:

- *partition_name* – the partition you are moving.
- *destination_segment_name* – a new or existing segment to which you are moving the partition. You cannot specify “default” as the *destination_segment_name*.

See the *Reference Manual: Commands*.

You must enable select into/bulkcopy to issue alter table ... move partition.

Locking

During a partition split, merge, or move, Adaptive Server takes an exclusive lock on the table on which it performs the operation, and the system table entries corresponding to the table.

Effect of split or merged partitions on indexes

Adaptive Server rebuilds all affected indexes when you perform a split, merge, or move partition on a table with indexes.

Table 3-1: Splitting and merging partitions on indexes

Command	Global nonclustered index	Local index	Local clustered index	Local nonclustered index
split partition	Index is rebuilt	All affected index partitions are rebuilt	Rebuilds all index partitions	Rebuilt on default segment
merge partition	No effect if the source and destination segments are the same	All affected index partitions are rebuilt	Rebuilds all index partitions	Rebuilt on default segment

If the table you are splitting or merging includes indexes on separate segments, the segments on which the newly rebuilt indexes reside depend on the type of index.

Table 3-2: Effect of split partition on index segments

Type of index	**After the split or merge operation**
Global nonclustered index	The index remains on the same segment as prior to the operation.
Local nonclustered index	• New index partitions (corresponding to the split or merge destination data partitions) are placed on the segment specified at the index level. • The indexes are placed on the default segment if you do not specify an index segment. • All unaffected index partitions (corresponding to other data partitions that were not involved in the split or merge) remain on the same segment as prior to the split or merge operation
Local clustered index	• New index partitions are placed on the same segment on which the corresponding destination data partition is placed. • The unaffected index partitions (corresponding to other data partitions not involved in the split or merge) remain on the same segment as prior to the split or merge operation.

Maintaining accurate statistics after a split or merge partition

Merging or splitting partitions removes statistics from systabstats and sysstatistics. Sybase recommends that you run update statistics after merging or splitting partitions.

CHAPTER 4 Maximum Size of Query in the Statement Cache

In versions earlier than 15.7 ESD #2, Adaptive Server had 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.

Adaptive Server 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.

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.

CHAPTER 5 Enhancements to *show_cached_plan_in_xml*

Adaptive Server 15.7 ESD #2 includes new information in the output of show_cached_plan_in_xml for:

- Index scan coverage
- Worktables used in the plan
- Dynamic partition elimination indication
- Total logical I/O (lio) and total physical I/O (pio)

Scan Coverage

The <scanCoverage> tag indicates whether the index scan for the query plan is “Covered” or “NonCovered”.

For more information on covered index scans, see “Indexes” in the *Performance and Tuning Series: Locking and Concurrency Control*.

Example

```
select show_cached_plan_in_xml(1139220075, 0)

<text>
<![CDATA[
SQL Text: select * from sysobjects group by name]]>
</text>
<IndexScan>
<VA>0</VA>
...
<scanCoverage> NonCovered </scanCoverage>
...
</IndexScan>
```

Worktables

The <wtObjName> tag under <WorkTable> provides the name of the worktable being used by the query plan. The <WorkTable> tag is provided under operators where the worktable is created.

Example

```
select show_cached_plan_in_xml(107219961, 0)
go

<text>
<![CDATA[
SQL Text: select distinct c1, c2 from t1, t2 where c1 = d1]]>
</text>
<opTree>
...
    <MergeJoin>
        ...
        <WorkTable>
            <wtObjName>WorkTable2</wtObjName>
    </WorkTable>
    ...
    </MergeJoin>
```

Dynamic partition elimination

show_cached_plan_in_xml shows dynamic partition elimination information within the “partitionInfo” section using the <dynamicPartitionElimination> tag, which indicates it is performing runtime partition elimination. show_cached_plan_in_xml indicates the compile time partition elimination under the <eliminatedPartition> tag.

For more information on partition elimination, see the chapter on parallel query processing in the *Performance and Tuning Series: Query Processing and Abstract Plans*.

Example 2

```
select show_cached_plan_in_xml(435436901,0)
go
<query>
   <statementId>435436901</statementId>
   <text>
```

```
      <![CDATA[

   . . .

                    <partitionInfo>
                        <partitionCount>3</partitionCount>
                        <eliminatedPartition>1</eliminatedPartition>
                        <eliminatedPartition>3</eliminatedPartition>
                        <dynamicPartitionElimination>No</dynamicPartition
                         Elimination>
                   </partitionInfo>

   . . .
```

Total logical I/O and total physical I/O

The <totalLio> and <totalPio> tags indicate the total logical I/O and total physical I/O per plan. There are two sets of <totalLio> and <totalPio> tags that indicate the actual and estimated values of the logical and physical I/O.

For more information on logical I/O and physical I/O, see “Displaying Query Optimization Strategies and Estimates” in the *Performance and Tuning Series: Query Processing and Abstract Plans*.

Example

```
select show_cached_plan_in_xml(1123220018, 0)
go
    <text>
           <![CDATA[
                SQL Text: select * from titles]]>
    </text>
<Plan>
<optTree>
. . .


            <est>
                <totalLio>3</totalLio>
                <totalPio>3</totalPio>
            </est>
            <act>
                <totalLio>3</totalLio>
                <totalPio>1</totalPio>
                </act>
  . . .
```

CHAPTER 6 Fast-Logged *bcp*

Adaptive Server version 15.7 ESD #2 and later allows you to fully log bcp running in fast mode, providing full data recovery. Earlier versions logged only page allocations.

See Chapter 4, "Transfer Data to and from Adaptive Server with bcp," in the *Utility Guide*.

CHAPTER 7

Enhanced parallel create index

Adaptive Server version 15.7 ESD 2 and later allows you to issue a parallel form of create index that uses the query execution engine to more efficiently execute the command.

Configuring enhanced parallel *create index*

Enhanced parallel create index is disabled by default, and is part of the enable functionality group configuration parameter. To enable Adaptive Server to use parallel create index:

1 Enable the enable functionality group configuration parameter:

```
sp_configure "enable functionality group", 1
```

2 Set the database option select into/bulkcopy/pllsort to true:

```
sp_dboption database_name, "select into", true
```

3 Set the following configuration parameter according to your hardware environment:

- number of worker processes – set to the maximum number of concurrently executing parallel threads used by all users
- max parallel degree – set to the maximum parallel degree used for an individual user, but not higher than number of worker processes
- max online engines – Sybase recommends that you configure a sufficient number of engines when configuring parallel threads, depending on hardware avaialbility and other workloads. number of worker processes is typically set higher than number of engines

Using enhanced parallel create index

Once Adaptive Server is configured for parallel create index, it determines if using parallel execution to execute create index is the best choice. If Adaptive Server determines that a serial query plan is the most efficient, it does not use a parallel query plan. If Adaptive Server determines that a parallel query plan is the most efficient, it selects an enhanced parallel query plan if:

- The table upon which the index is to be created:
  - Uses a data-only-locked format, and
  - Is not partitioned, and
  - The table is not empty
- The index you are creating is a non-clustered index, and
- The leading column of the index has at least two distinct values

Use the create index ... with consumers = *N* to force Adaptive Server to use parallel query plans when it would typically use a serial query plan. For example, Adaptive Server uses parallel query plans for the following even though if the table contains too few rows:

```
create index i1 on t1(c1, c3) with consumers = 3
```

If you use with consumers to force a parallel create index, and Adaptive Server does not select an enhanced parallel query plan, Adaptive Server uses a parallel create index query plan from a version of Adaptive Server earlier than 15.7 ESD #2.

Viewing parallel *create index* commands with showplan

If Adaptive Server is configured for parallel create index and chooses an enhanced parallel create index query plan, showplan displays information about the create index commands below the `PLL CREATE INDEX COORDINATOR` operator and `CREATE INDEX` operators. For example:

```
create index i1 on t5(c1) with consumers = 3
. . .

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

```
STEP 1
  The type of query is CREATE INDEX.

  5 operator(s) under root

|ROOT:EMIT Operator (VA = 5)
|
|   |PLL CREATE INDEX COORDINATOR Operator
|   |
|   |   |EXCHANGE Operator (VA = 3) (Merged)
|   |   |Executed in parallel by 3 Producer and 1 Consumer processes.

|   |   |
|   |   |   |EXCHANGE:EMIT Operator (VA = 2)
|   |   |   |
|   |   |   |   |CREATE INDEX Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator (VA = 0)
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  t5
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  Positioning at start of table.
|   |   |   |   |   |  Executed in parallel with a 3-way range repartitioning scan.
|   |   |   |   |   |  Using I/O Size 16 Kbytes for data pages.
|   |   |   |   |   |  With MRU Buffer Replacement Strategy for data pages.
```

CHAPTER 8 Precomputed Result Sets

A precomputed result set (PRS) is a view for which the result is computed, stored, and available for future use. Once configured for precomputed result sets, Adaptive Server precomputes queries and attempts to use the precomputed result during subsequent iterations. Precomputed result sets are also called materialized views.

Conceptually, a precomputed result set is both a view (because it includes query definition stored in the system tables) and a table (because it includes persistent data). You can perform many of the same operations that you perform on tables on precomputed result sets, including creating indexes and running update statistics.

Once Adaptive Server is configured to use precomputed result sets, the optimizer attempts to automatically rewrite each query using a precomputed result set. However, the final plan the optimizer selects is primarily cost based.

When the optimizer rewrites a query using a precomputed result set, it decides which precomputed result set is the best candidate. If the optimizer chooses to replace all, or part, of a query with a precomputed result set, it also adds any necessary compensation to the rewritten query (that is, any predicates needed to ensure the rewritten query is equivalent to the original user query). For example, if the user query includes a join of:

```
c1=c2 and c2=c3 and c3=c4
```

but the precomputed result set includes a join for:

```
c1=c2 and c3=c4
```

the rewritten query using the precomputed result set must have a compensation predicate similar to `c1=c3` to form an equivalent query.

Like an index, a precomputed result set has a maintenance cost for concurrent insert, update, and delete statements. Generally, precomputed result-set maintenance overhead consists of more than maintaining the indexes when the definition involves multiple table joins. Consequently, precomputed result sets are unsuitable for OLTP with heavy concurrent insert, update, and delete statements and simple index-based selects.

Benefits of precomputed result sets

Whether your site benefits from precomputed result sets depends on how they are designed. Although you may want to precompute as many queries as possible (particularly more joins) and make them available for multiple queries, precomputed result sets take extra disk space and have a higher maintenance cost. You can create extra indexes to help query performance, but these also incur an extra maintenance cost.

Precomputed result sets are best for frequently executed, expensive queries, such as those involving intensive aggregation and join operations. When you submit a query, the optimizer attempts to rewrite the query to use existing precomputed result sets instead of the base tables.

Generally, capture your application’s workload and design your precomputed result sets based on this workload. A good place to start is to create a combined join graph for all queries—along with their frequency of use—to indicate good candidates for using the same precomputed result set for multiple queries.

Test your precomputed result sets before putting them into production. If the queries are read-only or read-most, measure their performance gain against the extra disk space they use and the amount of time it takes them to populate the data; if it is a mixture of read-only or read-most, measure the impact of the precomputed result sets against the throughput.

Configuring Adaptive Server for precomputed result sets

Before you create or alter precomputed result sets, verify that these session set parameters are set correctly:

- set ansinull – on
- set arithabort – on
- set arithignore – off
- set string_rtruncation – on

Use create precomputed result set to create precomputed result sets. To use precomputed result sets for your queries, issue the set materialized_view_optimization command for the session.

Creating precomputed result sets

Use the create command to define precomputed result sets. The syntax is:

```
create {precomputed result set | materialized view}
     prs_name [(alternative_column_name
     [ [constraint constraint_name]
          unique (column_name,...)]

     [{immediate | manual} refresh]
     [{populate | nopopulate}]
     [enable | disable]
     [{enable | disable} use in optimization]
     [lock { datarows | datapages | allpages}]
     [on segment_name]
     [partition_clause]

as query_expression
```

You may specify the following in precomputed result sets:

- Partitions
- Segments
- Indexes (functional indexes not allowed)
- Unique keys (you must include the unique key constraint when you create precomputed result sets for immediate refresh)

If you drop the base table, the precomputed result set is changed to disabled.

See *Reference Manual: Commands*.

Identifying precomputed result sets

sp_help includes information about precomputed result sets in the Object_type and object_status columns:

```
sp_help mv1
Name  Owner  Object_type              Object_status
  Create_date
-----  -----  ----------------------  ------------------------------------
  --------------------
mv1   dbo   precomputed result set  immediate, enabled, enabled for QRW
   Apr 10 2012  8:57AM
...
```

sysobjects indicates an object is a precomputed result set with a value of RS in the type column.

Refreshing precomputed result sets

Precomputed result sets do not necessarily remain synchronized with the base tables from which they are constructed, and must also be refreshed, either automatically or manually. Configure the refresh policy when you create the precomputed result set, or later with the alter precomputed result set command:

- Immediate refresh – the precomputed result set is updated during the same transaction that updates the base tables. This is the default option. However, creating a precomputed result set for immediate refresh requires the user to own all the tables in the definition query.
- Manual refresh – the precomputed result set is updated with an explicit refresh command. Because manual refreshes are not maintained, Adaptive Server considers the data they contain to be stale (even immediately after you issue refresh), and selects these precomputed result sets for query rewrites only if the query is acceptable with stale data. The refresh command is executed under isolation level 1 or above.

The syntax to manually refresh a precomputed result set is:

```
refresh {precomputed result set | materialized view}
    [owner_name.]prs_name
```

If the schema of any of the base tables from which the precomputed result set is derived has changed, or if it was dropped and re-created (that is, the object ID has changed), the refresh command fails and returns an error indicating the precomputed result set must be dropped and re-created.

Only the owner of the precomputed result set can use the refresh command. If a user has permission to update the base table, he or she can also maintain the precomputed result set.

In most situations, the optimizer should use precomputed result sets with immediate refresh instead of manual refresh for query rewriting (unless you set materialized_view_optimization to stale).

Manually refreshing the precomputed result set is best when you control when insert, update, and delete statements occur. After they occur, perform a planned manual refresh of the precomputed result sets, then use the precomputed result sets to help your read-only applications. However, be aware of the time and extra disk space required to perform a manual refresh and plan accordingly.

Note After creating a precomputed result set, its owner may not have select permission on the base tables. If this occurs, manually refreshing the precomputed result set maintenance may fail, and it is not updated with the new changes from the base table.

You cannot execute the refresh command as part of a batch.

This example illustrates how to refresh a precomputed result set:

1 Create table t1:

```
create table t1 (
c1 int,
c2 int,
c3 char(5))
```

and populate it with this data:

```
 c1          c2          c3
----------- ----------- -----
          1           3  Aagg
          2           8  Xyz
```

2 Create table t2:

```
create table t2
(a1 int,
a2 int,
a3 char(5))
```

and populate it with this data:

```
 a1          a2          a3
----------- ----------- -----
          1           5  Ghr
          2           1  Gser
          3           6  agfh
```

3 Create the prs_1 precomputed result set:

```
create precomputed result set prs_1
```

```
unique (t1.c1, t2.a2)
as select t1.c1, t2.a2 from t1, t2 where t1.c1=t2.a1
```

prs_1 is created and populated with these initial rows:

```
 c1          a2
----------- -----------
          1           5
          2           1
```

4 If you insert the values 3, 7, and “fhi” into t1, prs_1 is immediately updated with the values 3 and 6:

```
 c1          a2
----------- -----------
          1           5
          2           1
          3           6
```

5 If you delete rows from t2 where a1 = 2, prs_1 is immediately updated with this change:

```
 c1          a2
----------- -----------
          1           5
          3           6
```

If Adaptive Server rolls back the transaction updating the base table, it also rolls back the immediate update on the base table’s precomputed result set as part of the same transaction.

Altering precomputed result sets

Use the alter command to change the precompute result set’s policies or properties. The syntax is:

```
alter {precomputed result set | materialized view}
        prs_name
    {immediate | manual} refresh
    | enable | disable
    | {enable | disable} use in optimization
```

See *Reference Manual: Commands*.

This example alters the author_prs precomputed result set from manual to immediate:

```
alter precomputed result set author_prs
immediate refresh
```

alter automatically refreshes the precomputed result set when you change from a manual to an immediate refresh, or from disable to enable. Altering a precomputed result set for disable use in optimization prevents the precomputed result set from participating in future query rewriting. However, any plans already cached using the precomputed result set are not recompiled.

Similar to other DDL commands, you cannot issue alter precomputed result set as part of a multistatement transaction (unless you set the ddl in tran option to true for the database). You must be the owner of the precomputed result set to issue alter precomputed result set

If the base table or view on which the precomputed result set is based is dropped or altered, the precomputed result set is automatically altered to disable. Adaptive Server sets the precomputed result sets to disable when you run bcp in or select into existing against any base table from which the precomputed result set is generated.

When you alter a precomputed result set to disable (with the alter precomputed result set command or with the alter table command on the base table), any plans already cached that use the precomputed result set are recompiled when the plan is next executed.

Dropping or truncating precomputed result sets

Dropping a precomputed result set deletes its data, removes any system table entries, and deletes the precomputed result set. The syntax is:

```
drop {precomputed result set | materialized view}
    prs_name
```

You must be the owner of the precomputed result set to issue drop precomputed result set. See *Reference Manual: Commands*.

This example drops authors_prs:

```
drop precomputed result set authors_prs
```

Use the truncate command to truncate data in a precomputed result set. truncate retains the definition of the precomputed result set in the system table, ensuring that the precomputed result set can be later repopulated using the refresh command.

Truncating a precomputed result set moves it to a disabled state. Adaptive Server moves the precomputed result set back to the enabled state when you issue refresh prs.

The syntax is:

```
truncate {precomputed result set | materialized view}
     prs_name
```

This example truncates the author_prs:

```
truncate precomputed result set authors_prs
```

Adaptive Server implements the refresh command first as a truncate command and then recomputes the precomputed result set. In the unlikely event that the truncate command succeeds but the recompute fails, the precomputed result set is left disabled and you may reissue the refresh command.

Configuring the staleness

Precomputed result sets rely on updates from their base tables to ensure data is current. When a precomputed result set is configured for immediate updates, any base table updates also update the precomputed result set. This update occurs as an incremental maintenance using changes to the base tables. However, if a precomputed result set is configured for manual updates, data may become stale because updates only occur when you run the refresh command (during which Adaptive Server recomputes the precomputed result set instead of performing an incremental maintenance).

Unless you specify otherwise, Adaptive Server does not use a stale precomputed result set to rewrite queries. Use the set materialized_view_optimization to specify at the session level whether Adaptive Server can use stale precomputed result sets when rewriting queries during optimization:

```
set materialized_view_optimization {disable | fresh | stale}
```

For Adaptive Server to use stale precomputed result sets to rewrite queries:

- The user must be the owner of the stale precomputed result set, and

- set materialized_view_optimization must be set to stale.

See *Reference Manual: Commands*.

Querying precomputed result sets

Adaptive Server allows you to select information from precomputed result sets, but you cannot insert, update, or delete information from the precomputed result set. Instead, you must insert, update, or delete information from the base tables, and then refresh the precomputed result set.

Rewriting queries

Query rewrite mechanisms generate alternative plans based on available precomputed results. The alternative plans compete with other plans in the optimizer, and Adaptive Server selects the one with the lowest estimated cost. However, the query rewrite mechanism works only with select queries; it does not consider insert, update, delete, and select into queries for rewrite.

Adaptive Server may rewrite an entire query to create an equivalent precomputed result set, or it may rewrite part of a query, depending on the query properties and the available precomputed result sets. The precomputed result set must completely cover the logical data set for queries that Adaptive Server rewrites.

For example, if you have a query similar to this, which is complicated and involves multitable joins and many predicates, groupings, and aggregations:

```
select t1.col1,t2.col1,t3.col1,
     sum(t1.col3),sum(t2.col3), sum(t3.col3)
from t1, t2, t3
where t1.col1 = t2.col1
     and t2.col1 = t3.col1
     and t1.col2 < 60
     and t1.col1 > 5
     and t1.col2 + t2.col2 < 40
group by t1.col1, t2.col1, t3.col1
```

And create this precomputed result set:

```
create precomputed result set newprs
as
select t1.col1 as p11, t1.col2 as p12, t2.col1 as p21,
     t2.col2 as p22, t3.col1 as p31, t3.col2 as p32,
     sum(t1.col3) as agg_s13,sum(t2.col3) as agg_s23,
     sum(t3.col3) as agg_s33
from t1, t2, t3
where t1.col1 = t2.col1
and t1.col2 < 60
and t1.col2 + t2.col2 < 40
group by t1.col1, t2.col1, t3.col1, t1.col2,
     t2.col2, t3.col2
```

The query rewrite mechanism may alter the original query to something similar to this, which is much simpler and may be cheaper to execute:

```
select p11,p21,p31,
     sum(agg_s13),sum(agg_s23),sum(agg_s33)
from newprs
where p21 = p31
     and p11 > 5
group by p11, p21, p31
```

Updating statistics

Adaptive Server allows you to run updates statistics on precomputed result sets.

Replicating precomputed result sets

These precomputed result set commands are replicated:

- create precomputed result set
- alter precomputed result set
- drop precomputed result set
- truncate precomputed result set
- refresh precomputed result set

Although precomputed result set DDLs can be replicated, precomputed result sets cannot be marked for replication. That is, unlike a regular table, Adaptive Server does not replicate any maintenance changes occurring on data stored in a precomputed result set, regardless of whether the changes are initiated from a precomputed result set DDL (which itself could be replicated), or are part of the base table update transaction (initiated from an immediate refresh policy).

Precomputed result set DDL replication is supported only between two Adaptive Servers that include the precomputed result set functionality.

Restrictions

Precomputed result sets cannot include:

- References to another precomputed result set. However, precomputed result sets can reference a view if you create the precomputed result set with a manual refresh policy.
- Expressions in the select list. However, you can include an expression as part of the group by list.
- Encrypted columns, or result sets that encrypt their own column data.
- References to tables or functions in another database.
- References to virtual computed columns in a base table. However, precomputed result sets can refer to materialized computed columns in a base table.
- compute, compute by, group by all, or order by clauses.
- Nondeterministic functions (for example, getdate).
- XML.
- Subqueries.
- Outer and semijoins.
- Calls to user-defined functions.
- Derived tables.
- Reference to system, temporary, or fake tables.
- Union clauses.
- Any constraint other than unique key constraints.

- Columns that are defined with identity, null, or not null clauses.
- Defaults or rules.
- Cursors.
- Statistical aggregate functions.
- text, image, or unitext columns.

In addition to the previous restrictions, precomputed result sets using the immediate refresh policy cannot include:

- top, min, max, and avg commands
- distinct clauses
- Self joins
- Functions
- References to proxy tables
- References to a view
- having clauses that reference the result of an aggregate
- sum functions that reference a nullable expression

CHAPTER 9 Concurrent *dump database* and *dump transaction* Commands

Adaptive Server versions 15.7 ESD #2 and later allow 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.

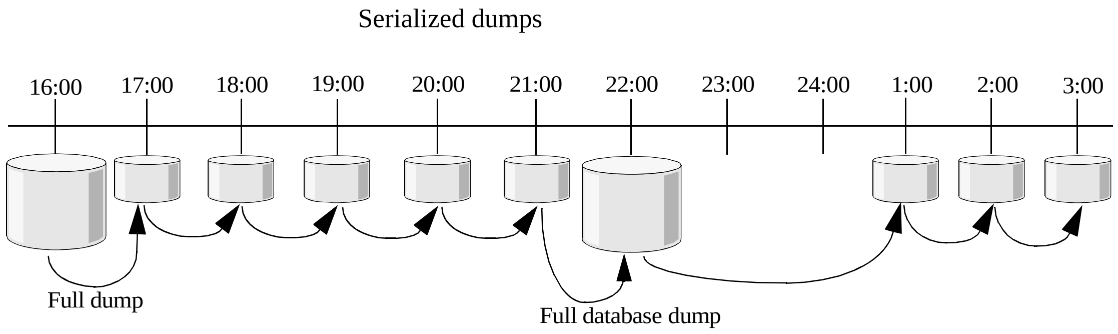
dump database works in two phases: the first phase copies the database pages to the dump archive, and the second phase copies the active part of the transaction log to the dump archive. dump transaction uses a single phase when it copies the active part of the transaction log to the dump archive.

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.

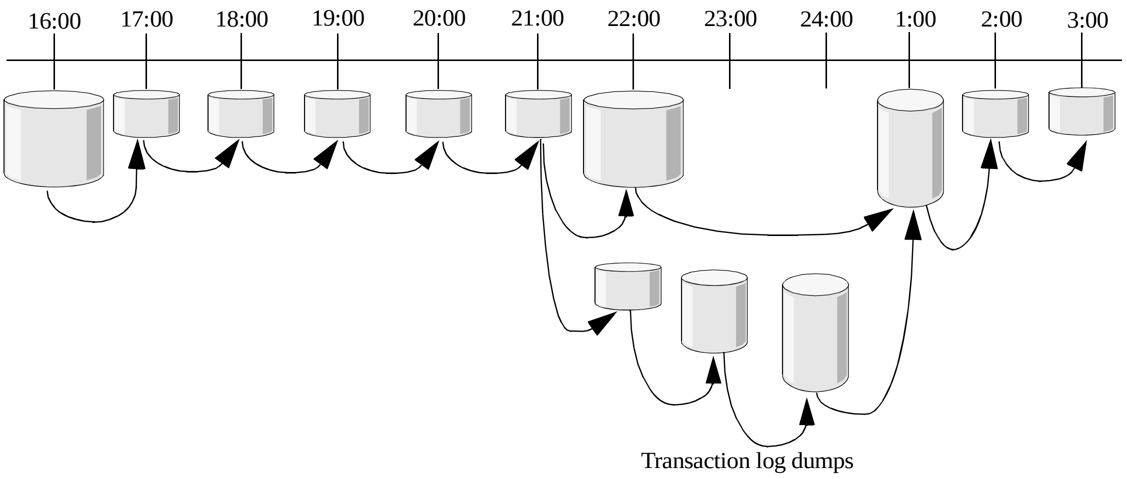
Any dump transaction that occurs concurrently with dump database (that is, completes before dump database starts copying the active log), cannot be loaded on top of that database dump: the dump of the transaction log belongs to the preceding load sequence.

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. See “Dump history file” on page 54.

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.

Configuring Adaptive Server to use concurrent dumps

Enable Adaptive Server to perform concurrent dumps with the enable concurrent dump tran configuration parameter.

enable concurrent dump tran

Summary information	
Default value	0 (off)
Valid values	0 (off), 1 (on)
Status	Dynamic
Display level	Comprehensive
Required role	System administrator
Configuration groups	Application Functionality

enable concurrent dump tran enables or disables Adaptive Server to use concurrent dumps.

enable concurrent dump tran is part of the enable functionality configuration parameter group. The default value for the parameters in this group depends on the value to which enable functionality group is set. A value of `DEFAULT` for the individual configuration parameters in this group—other than enable functionality group—means they are set to the same value as enable functionality group. That is, if you set enable functionality group to 1, a value of `DEFAULT` for any other configuration parameter in the group is 1.

Aside from the value for enable functionality group, you can ignore values of `DEFAULT` in the output from sp_configure and sp_helpconfig for individual configuration parameters in the Application Functionality group.

See the *System Administration Guide, Volume 1*.

Restrictions

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.

CHAPTER 10 Hash-Based Update Statistics

Adaptive Server versions 15.7 ESD #2 and let you gather hash-based statistics on minor index attributes and unindexed columns instead of using sort-based statistics. Using hash-based instead of sort-based statistics improves performance by reducing the number of required scans, and by avoiding disk-based sorting.

Hash-based statistics allow greater flexibility than sort-based statistics:

- Running hash-based statistics should require less time to run sort-based statistics, increasing the amount of work you can accomplish during a maintenance window.
- Because hash-based statistics require less procedure cache, you may be able to run update statistics on data-only-locked tables outside the maintenance window since the tempdb buffer cache, which usually uses the default data cache, is typically much larger than the procedure cache.
- Generally, hash-based statistics do not require large tempdb disk allocations. If you previously increased the size of tempdb to accommodate large sorts from update statistics, you may be able to redeploy this space.
- update index statistics and update all statistics with hashing may run faster than sorting with sampling. However, an exception may be update statistics *table_name*(*col_name*).

Hash-based statistics use a low-domain algorithm for columns with fewer than 65536 unique column values, and a high-domain algorithm for columns with 65536 or more unique column values. Of the two algorithms, low-domain hashing produces the more accurate histogram because Adaptive Server uses the actual counts of all the domains values to create the histogram. High-domain hashing may produce a less accurate histogram because Adaptive Server produces an in-memory intermediate histogram that it updates for each block of 65536 unique values.

Because gathering hash-based statistics is CPU-intensive, you may want to create an execution class with EC3 attributes to which you can assign the update statistics login. Adaptive Server gives lower priority to update statistics maintenance sessions, reducing the impact when the maintenance window is small or nonexistent.

When running update statistics, Sybase recommends that you:

- Use partitioned tables so only the active partitions require update statistics maintenance
- Use datachange to determine when to run update statistics
- Avoid running update statistics on allpages-locked tables when concurrency may be an issue (since update statistics uses level 1 page locking on allpages-locked tables, which has less concurrency than the dirty reads on data-only-locked tables)

Enabling hash-based statistics

Use the update statistics hashing configuration parameter to enable Adaptive Server to gather hash-based statistics.

update statistics hashing

Summary information	
Default value	partial
Range of values	One of: • off – no hashing. • on – hashing on all columns. • partial – hashing only for low unique count columns. • default – off .
Status	Dynamic
Display level	Comprehensive
Required role	System administrator
Configuration group	General Information

update statistics hashing enables Adaptive Server to gather hash-based statistics.

Gathering hash-based statistics

Use this syntax to create hash-based statistics:

```
update index statistics
     table_name [[partition data_partition_name] |
     [ [index_name [partition index_partition_name]]]
     [using step values]
     [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
         [, max_resource_granularity = N [percent]]
         [, histogram_tuning_factor = int ]

update all statistics
     table_name [partition data_partition_name]
     [using step values]
     [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
         [, max_resource_granularity = N [percent]]
         [, histogram_tuning_factor = int ]

update statistics
     table_name [[partition data_partition_name]
          [ (col1, col2, …) | (col1), (col2), …] |
     [ [index_name [partition index_partition_name]]]
     [using step values]
     [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
         [, max_resource_granularity = N [percent]]
         [, histogram_tuning_factor = int]
```

Determine the level of hashing with [no_hashing | partial_hashing | hashing]:

- no_hashing – uses the sort algorithm from versions of Adaptive Server earlier than 15.7 ESD #2.
- partial_hashing – Adaptive Server uses hashing for low unique count domains. If Adaptive Server encounters unique column counts that exceed the 65536 threshold, it uses an extra scan with a sort. Sorts are also used if the previous histogram generated on that column indicated that the number of unique values was greater than or equal to 65536.

The default for these parameters is the configured value for update statistics hashing.

See *Reference Manual: Commands*.

This example gathers hash-based statistics for the authors table:

```
update index statistics authors with hashing
```

Explicitly specifying hash-based statistics in an update statistics command takes precedence over the value for the update statistics hashing configuration parameter. In the example above, update statistics ... with hashing takes precedence over the server-level update statistics hashing parameter.

You cannot use the consumer and sampling parameters when you gather hash-based statistics. Adaptive Server attempts to use low-domain hash-based statistics with partial_hashing before defaulting to the sort-based statistics, which support the consumer and sampling parameters.

Hash-based update statistics allow you to specify column sets as a comma-separated list of column names in parentheses:

update statistics *table_name* (*column1*), (*column2), (column3*), ...

This syntax allows Adaptive Server to perform a single table scan that updates statistics on all columns. However, Adaptive Server issues an error message if you include multiple columns within the same parentheses to gather hashing statistics.

Setting the distribution granularity

Use the histogram tuning factor configuration parameter to determine a histogram's distribution granularity (that is, the step count) for the server.

The update statistics ... histogram_tuning_factor parameter determines update statistics' distribution granularity for histograms, which helps to isolate skew values and improves equi-weight range cell behavior. update statistics transforms the final histogram to conform to the configured step count for range cells, but retains any frequency cells. If the final histogram update statistics produces appears to skew the weights, try increasing the tuning factor to create more equi-weight range cells.

Setting the buffer manager memory

Hashing can use a significant amount of tempdb buffer cache memory. By default, update statistics uses the value to which the max resource granularity configuration parameter is set, which is the percent of the tempdb buffer cache that can be used.

See "Setting Configuration Parameters" in the *System Administration Guide, Volume 1*.

Limit the amount of buffer memory used with update statistics ... max_resource_granularity. If Adaptive Server reaches this value, it selects a column from which it recycles the memory so it can finish hashing the remaining columns. The histograms for columns from which it recycles resources are gathered on subsequent scans using hashing. To avoid the extra scans, increase the value for max resource granularity, if necessary.

CHAPTER 11

Including progress messages with update statistics

update index statistics, update statistics, and update all statistics for Adaptive Server versions 15.7 ESD #2 and later include the print_progress parameter, which allows these commands to display progress messages.

Using the print_progress parameter

The syntax for print_progress is:

- update index statistics:

```
update index statistics
table_name [[partition data_partition_name] |
. . .
[, print_progress = int]
```

- update all statistics:

```
update all statistics
table_name [partition data_partition_name]
. . .
[, print_progress = int]
```

- update statistics

```
update statistics
table_name [[partition data_partition_name]
. . .
[, print_progress = int]
```

Where:

- 0 – (the default) disables print_progress to not display any progress messages
- 1 – enables print_progress to display progress messages

This example shows the progress messages when update statistics is run on table bigtable:

```
update statistics bigtable with print_progress=1
Update Statistics STARTED.
Update Statistics index scan started on index 'bigtable_NC1'.
Update Statistics table scan started on table 'bigtable' for summary statistics.
Update Statistics FINISHED.
```

This example shows the progress messages when update index statistics is run on table bigtable:

```
update index statistics bigtable with partial_hashing, print_progress=1
Update Statistics STARTED.
Update Statistics index scan started on index 'bigtable_NC1'.
...It is using existing index scan to hash minor column 'a2' (column id = 2).
...Column 'a2' (column id = 2) is moved from hashing to sorting.
Update Statistics table scan started on table 'bigtable' for summary statistics.
Update Statistics table scan started on table 'bigtable'.
...Sorting started for column 'a2' (column id = 2).
Update Statistics FINISHED.
```

This example shows the progress messages when update statistics ... with hashing is run on table bigtable:

```
update statistics bigtable (a1), (a2), (a3) with hashing, print_progress=1
Update Statistics STARTED.
Update Statistics table scan started on table 'bigtable'.
...Column 'a3' (column id = 3) is picked as hash victim due to limited resource.
Update Statistics table scan started on table 'bigtable'.
```

CHAPTER 12 Enhancements to Dump and Load

Adaptive Server 15.7 ESD #2 includes enhancements to the dump and load commands:

- The dump configuration command allows you to back up the Adaptive Server configuration file, the dump history file, and the cluster configuration file. See the *Reference Manual:Commands*.
- Adaptive Server 15.7 ESD #2 introduces dump configurations that define the options to create a database dump. Backup Server then uses the configuration to perform a database dump. You can use:
  - The dump configuration to create, modify, or list dump configurations, then use dump database or dump transaction with the dump configurations to perform dumps. See the *Reference Manual:Commands*. See the *Reference Manual:Procedures* for information about sp_config_dump.
  - The enforce dump configuration configuration parameter to enable dump operations with dump configurations. See "Configuration parameters" on page 50.
  - The configuration group "dump configuration" that represents user-created dump configurations. See "Using dump configurations" on page 52.
- Dump history – Adaptive Server 15.7 ESD #2 allows you to:
  - Preserve the history of dump database and dump transaction commands in a dump history file that Adaptive Server can later use to restore databases up to a specified point in time. See "Dump history file" on page 54.
  - 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
    ```

    For details on extensions to load database, see the *Reference Manual:Commands*.

- Use the sp_dump_history system procedure to purge dump history records.

  For details on sp_dump_history, see the *Reference Manual:Procedures*.

- 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.

- The dump with listonly command has been enhanced in Adaptive Server 15.7 ESD #2 to provide two options. You can:

  - 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.

  For details on extensions to load database and load transaction, see the *Reference Manual:Commands*.

  For details on the enhanced dump header, see "Enhancements to dump header" on page 55.

Configuration parameters

Adaptive Server includes these configuration parameters for dump configuration.

enforce dump configuration

Summary information	
Default value	0 (disabled)
Range of values	0 (disabled), 1 (enabled)

Summary information	
Status	Dynamic
Display level	Basic
Required role	System administrator
Configuration groups	Backup/Recovery

enforce dump configuration determines if Adaptive Server uses a dump configuration to perform database dumps.

When enabled, Adaptive Server allows dump operations only with dump configurations. If the dump command has specified parameters, such as blocksize, compression, and so on, those specified values do not override the values defined by the dump configuration.

When disabled, Adaptive Server uses the parameter values specified on the command line, and overrides the values defined by the dump configuration.

enable dump history

Summary information	
Default value	0 (disabled)
Range of values	0 (disabled), 1 (enabled)
Status	Dynamic
Display level	Basic
Required role	System administrator
Configuration groups	Backup/Recovery

dump history update determines whether there are updates to the dump history file at the end of the database dump operation.

By default, Adaptive Server updates the dump history file after every database dump.

dump history filename

Summary information	
Default value	dumphist
Range of values	

Summary information	
Status	Dynamic
Display level	Basic
Required role	System administrator
Configuration groups	Backup/Recovery

dump history filename specifies the path of your dump history file.

Using dump configurations

The dump configuration configuration parameter group represents these user-created dump configurations:

- stripe directory – is the directory in which files are archived during the dump operation. Archived files are typically named using this convention:

  database_name.nump_type.date-timestamp.stripeID

- external api name – is the name of the external API (byte stream device) to be used for the dump operation, and must conform to this format:

  ```
  External API Name::Options
  ```

- number of stripes – is the number of stripe devices to use during the dump operation. By default, a single stripe device is used.
- number of retries – is the number of times the server tries the dump operation for nonfatal errors up to a maximum of 5 times. The default is 0.
- block size – is the block size for the dump device and overrides the default block size for the device. blocksize must be at least 1 database page, and an exact multiple of the database page size.
- compression level – is the compression level for compressed dumps. By default, compression is disabled.
- retain days – is the number of days during which the dump cannot be overwritten. Backup Server requires confirmation to overwrite an unexpired volume. By default, retaindays is 0, and dumps can be overwritten.
- init – specifies whether the volume must be reinitialized. The default is “noinit”.

- verify – specifies if Backup Server must perform a minimal page-header or full structural row check on the data pages as they are copied to archives. There is no structural check made to global allocation map (GAM), object allocation map (OAM), allocation pages, indexes, text, or log pages. By default, there is no verification of data pages durig archiving.
- notify – the default message destination to Backup Server. One of:
    - client - route messages to the terminal that initiated the dump command.
    - operator_console - route messages to the terminal on which Backup Server is running
- remote backup server name – specifies the remote Backup Server used for a dump operation. The default is SYB_BACKUP.

Examples

Example 1 Contains multiple dump configurations as created in the Adaptive Server configuration file:

```
[dump configuration : dmp_cfg1]
   stripe_dir = /work/dmp_cfg1_dir
   ext_api = DEFAULT
   num_stripes = 5
   retry = 0
   blocksize = DEFAULT
   compression = 9
   retaindays = DEFAULT
   init = DEFAULT
   verify = DEFAULT
   backup_srv_name = DEFAULT

[dump configuration : dmp_cfg2]
   stripe_dir = /work/dmp_cfg2_dir
   ext_api = syb_tsm
   num_stripes = DEFAULT
   retry = 3
   blocksize = DEFAULT
   compression = DEFAULT
   retaindays = DEFAULT
   init = DEFAULT
   verify = DEFAULT
   backup_srv_name = SYB_REMOTE
```

Dump history file

Adaptive Server maintains the history of successful and failed backups from dump database and dump transaction commands in a dump history file. Adaptive Server 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 Adaptive Server instance has a dump history file with information about all database dumps and server configuration dumps, successful or not. The default location of this file is the location specified with the -m startup parameter, or the $SYBASE directory if -m is not specified.

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
- Adaptive Server 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 Adaptive Server. The user starting Adaptive Server requires appropriate read and write permissions to the dump history file.

Enhancements to dump header

Adaptive Server versions 15.7 ESD #2 and later store information about database devices 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

CHAPTER 13 Dropping Columns from a Table Without Performing a Data Copy

The no datacopy parameter to the alter table drop column allows you to drop columns from a table without performing a data copy, and reduces the amount of time required for alter table drop column to run.

The syntax is:

```
alter table [[database.][owner].table_name
    {add column_name datatype}
. . . }

    modify column_name
    drop {column_name [, column_name]...
        with exp_row_size=num_bytes
            | transfer table [on | off]}
            | no datacopy
```

Instead of immediately removing the columns from the table, no datacopy updates the system tables, indicating the affected rows will be reformatted the next time you run reog rebuild or another datacopy operation (the space allocated for dropped columns (including large objects) is not freed until the next time you run reorg rebuild).

This example drops the total_sales column from the titles table without a data copy:

```
alter table titles
drop total_sales
with no datacopy
```

Restrictions

You cannot use the no datacopy parameter on:

- Materialized or virtual computed columns
- Encrypted columns

- XML columns
- IDENTITY columns
- Java columns
- Proxy tables
- Columns using these datatypes:
  - timestamp
  - bit
- You cannot change the locking scheme of a table:
  - That has been affected by a no datacopy operation
  - For which you have not yet executed a reorg rebuild or datacopy operation since the last drop column with no datacopy

  You must run reorg rebuild before changing the locking scheme of a table.

CHAPTER 14 Expanded Maximum Database Size

Adaptive Server versions 15.7 ESD #2 and later expand the maximum size of a database to approximately 64 terabytes by converting the logical page number from a signed integer to an unsigned integer.

Versions of Adaptive Server earlier than 15.7 ESD #2 allowed for a maximum database size of approximately 32 terabytes.

The maximum size of a database depends on its logical page size:

- 2K page server = 8TB
- 4K page server = 16TB
- 8K page server = 32TB
- 16K page server = 64TB

Note Because Adaptive Server reserves 256 logical page IDs—which cannot be allocated or used—at the high end of the logical page range, the sizes listed above are slightly higher than the actual amount of space available for use. This overhead reduces the actual amount of space available by 256 times the logical page size for each listed page size (for example, the actual available size for a 2K server is 8TB – (256 x 2K)).

Expanding the maximum size of a database requires you to change the datatypes for these columns from int to unsigned int:

- sysusages – lstart, size, and unreservedpgs
- sysaltusages – lstart and size
- syspartitions – firstpage, rootpage, dataoampage, and indoampage
- systabstats – leafcnt, pagecnt, emptypgcnt, warmcachepgcnt, unusedcnt, and oampgct
- syslocks – page
- syslogshold – page

- systhresholds – free_space

Note You must change the expected result for any queries that rely on the columns listed above from an int to an unsigned int.

These functions now return an unsigned int result instead of an int:

- curunreservedpgs
- used_pages
- data_pages
- reserved_pages
- lct_admin

CHAPTER 15 User-Defined Optimization Goal

Adaptive Server versions 15.7 ESD #2 and later allow you to create user-defined optimization goals (a set of all active optimizer criteria), which allow you to:

- Create a new optimizer goal
- Define a set of active criteria included in the goal
- Activate the goal for a server, session, procedure, and query level
- Dynamically change the goal content without disconnecting and reconnecting the client session

Once you create the user-defined optimization goals, you can invoke them at the server level or for a user session.

Creating a user-defined optimization goal

Use sp_optgoal to create a user-defined optimization goal. The syntax is:

```
sp_optgoal "goal_name", "save"
```

where:

- *goal_name* – which cannot be longer than 12 characters, is the name of the goal you are creating.
- save – creates the goal if it does not already exist

See the *Reference Manual: Procedures*.

Example

This example creates a goal called goal_1571, which:

1. Sets the optimization level to ase157ga
2. Sets the optimization goal to allrows_mix
3. Enables hash joins
4. Enables the optimization criteria for CR # 123456

5 Disables the optimization criteria for CR # 234234:

```
set plan optlevel ase157ga
set plan optgoal allrows_mix
set hash_join 1
set CR123456 1
set CR234234 0
go
execute sp_optgoal "goal_1571", "save"
go
```

Setting the goal server-wide and for the session

Use the sp_configure 'optimization goal' parameter to set a goal to apply server wide. The syntax is:

sp_configure 'optimization goal',1,'*goal_name*'

For example, to set the goal_1571 for the server, enter:

```
sp_configure 'goal',1,'goal_1571'
```

Use set to set the goal for the current session or server-wide. The syntax is:

set plan optgoal *goal_name*

For example, to set goal_1571 for the current session:

```
set plan optgoal goal_1571
```

This example uses *goal_name* in an abstract plan at the query level:

```
select count(*) from tab1,tab2
PLAN '(use optgoal goal_1571)'
go
```

Reporting on goals

sp_optgoal 'show','goal_name' reports all individual criteria activated by the goal named *goal_name*. For example:

```
sp_optgoal 'goal_1571', 'show'
```

sp_optgoal @@optgoal, 'show' reports current goal settings:

```
sp_optgoal @@optgoal, 'show'
name
-----------------------------------------------------------------------------
distinct_sorted
distinct_sorting
distinct_hashing
group_sorted
group_hashing
nl_join
merge_join
append_union_all
merge_union_all
merge_union_distinct
hash_union_distinct
opportunistic_distinct_view
parallel_query
order_sorting
store_index
replicated_partition
ndex_union
streaming_sort
nary_nl_join
alternative_greedy_search
cr562947: OPTLEVEL EXCEPTION SEE CR - allow cursor table scans
data_page_prefetch_costing: clustered row bias added
mru_buffer_costing: wash size buffer limit for MRU
cr546125: implicitly updatable cursor non-unique index scan
cr545771: improves multi-table outer-join and semi-join costing
cr545653: avoid inner table buffer estimate starvation
cr545585: covered iscan CPU costing too expensive
cr545379: disallow reformatting on user forced index scan
cr545180: avoid reformat with no sargs if useful index exists
cr545059: reduce usage of buffer manager optimization sorts
cr544485: mark subquery join predicates with distinct view as sargs
cr534175: compute GROUP BY worktables in nested subqueries only once when possi
ble
cr531199: increases the number of useful nested loop join plans considered
cr500736: supports nocase sortorder columns in mergejoin and hashjoin keys
cr487450: improves DISTINCT costing of multi-table outer joins and/or semi-joins
cr467566: allow abstract plans and statement cache to work together
cr497066: infer the nullability of isnull() by looking at its parameters
cr421607: support NULL=NULL merge and hash join keys
cr552795: eliminate duplicate rows during reformatting when they're not needed
imdb_costing: 0 PIO costing for scans for in-memory database
allow_minmax: allow local session to consider MINMAX optimization
cr646220: enable better store index key generation with correlated predicate
```

CHAPTER 16 Shared Query Plans

Adaptive Server versions 15.7 ESD #2 and later allow you to share query plans, avoiding the need for Adaptive Server to create or recompile query plans that are identical to existing plans. Shared query plans are cloned from primary query plans under concurrent system.

Use sp_configure to enable Adaptive Server to use shared query plans. The syntax is:

```
sp_configure 'enable plan sharing', 1
```

enable plan sharing is part of the enable functionality group. See the *System Administration Guide, Volume 1* for information about setting parameters that are part of this group.

You should see a performance improvement as Adaptive Server shares query plans instead of reusing or recompiling them. You may see a slight change to procedure cache memory usage as primary query plans are pinned in the cache while Adaptive Server uses their shared query plans.

For a query plan to be shareable:

- It must be a lightweight procedure (LWP), either:
  - For dynamic SQL, (often used for ODBC/JDBC prepared statements), or:
  - Resulting from the operation of the statement cache, where qualifying statements are transformed into LWPs for reuse
- It is a lava query plan
- It cannot include instead-of-tiggers
- It may include only:
  - declare, insert, delete, update, merge, or select statements
  - A single statement, unless the first statement is a declare statement, in which case the query plan may include two statements
  - Serial query plans
  - It cannot access plans that reference Java objects

The show_cached_plan_in_xml function describes the sharing of plans under the <planSharing> element.

- `shareable` – plan may be shared.
- `notShareable` – plan may not be shared.
- `primary` – primary plan (clones of which are being shared).
- `shared` – shared plan (clone of a primary plan).

CHAPTER 17 Initializing databases asynchronously

The async_init parameter for the alter database and create database commands lets you asynchronously initialize a database while it is being used. That is, the database is immediately available when it is created or altered, not when the database initialization is complete. The initialization is transparent to the user.

Any task that uses a page of the database that is not yet initialized performs an initialization of the allocation unit on which the page resides.

The asynchronous initialization is performed by a service task that is started by the create database or alter database command. When it restarts, Adaptive Server automatically starts a new service task that completes the initialization. In a clustered environment, if an instance running the service task fails or is shut down, the coordinating instance starts a new service task to complete the initialization.

Configuring Adaptive Server to asynchronously create or alter databases

The enable async database init configuration parameter determines whether Adaptive Server asynchronously creates or alters databases.

enable async database init

Summary information	
Default value	0 (off)
Valid values	0 (off), 1 (on)
Status	Dynamic
Display level	Comprehensive
Required role	System administrator

Summary information	
Configuration groups	SQL Server Administration

enable async database init ensures that all create database and alter database commands initialize databases asynchronously by default.

Creating or alter databases asynchronously

The syntax to create databases asynchronously is:

```
create [temporary] database database_name
    [on {default | database_device} [= size]
. . .
    [with {override
    | default_location = "pathname" [,[no]async_init] }
    [for {load | proxy_update}]
```

noasync_init indicates the database is initialized synchronously.

The syntax to alter a database asynchronously is:

```
alter database database_name
    [on {default | database_device } [= size]
. . .
    [with override [,[no]async_init]]
    [for load]
    [for proxy_update]
```

noasync_init indicates that you are extending a database, and that Adaptive Server initializes the extended space synchronously.

Using the [no]async_init parameter for create or alter database overrides the settings for enable async database init.

Determining if there is space to be initialized

Adaptive Server synchronizes a portion of the data and log segments synchronously before making the database available, allowing the initializer to work ahead of any commands that require space in the database. However, you may occasionally see a performance impact to commands normally run against the database while Adaptive Server is busy initializing the space. This occurs because a command that requires space that is not yet initialized must initialize the space before it proceeds.

Information about initialized space is stored in sysattributes.

To determine if there is space not yet initialized in the database (for example, if the initializer terminated prematurely and left part of the database uninitialized), issue a query similar to:

```
select lstart=object_info1, size=object_info2, segmap=object_info3
from master..sysattributes where class=42 and object=db_id("mydb")
lstart           size             segmap
 -----------     -----------  -----------
       1536         3584000            3
       5120           51200            4
```

If the query returns one or more rows, the database contains space not yet initialized (in this query, the mydb database). This query does not indicate if the asynchronous initialization service task is running, only that it is not finished (if it was finished, the result set would contain zero rows).

Use a query similar to the following to determine if the initializer is running on a specific database (in this query, the test database):

```
select spid from sysprocesses
where dbid=db_id("test") and cmd="CRDB AUINIT"
 spid
------
    22
```

Adaptive Server prints this message to the error log once the asynchronous initialization service task is running:

```
Asynchronous initialization of database 'database_name'
has completed.
```

If the asynchronous initialization service task stops prematurely, Adaptive Server prints this message to the error log:

```
Asynchronous database initialization terminated
prematurely for database '%.*s'. Use DBCC
```

```
DBREPAIR(%.*s, async_database_init, start) to restart
it if required as uninitialized pages will incur a small
performance penalty when they are first referenced.
```

Restrictions

- You cannot initialize these databases asynchronously, even if you explicitly use the async_init parameter:
  - All system databases
  - All temporary databases, system or user
  - Archive databases
  - Proxy databases
  - Any database created with the for load option
- These commands cannot be run in a database that it still undergoing initialization:
  - unmount database
  - alter database ... log off
- You can put the database into single user mode during initialization. However, the initializer does not run while the database is in single user mode, and is automatically restarted to continue initialization when you take the database out of single user mode.

Note You may notice a slight performance impact to DMLs that use the space in the database being initialized while the asynchronous initialization service task is running.

CHAPTER 18 In-Row Large Object Compression

Adaptive Server versions 15.7 ESD #2 and later support in-row large object (LOB) compression. See the *Compression Users Guide*.

Adaptive Server uses in-row LOB compression if:

- The table is implicitly or explicitly row- or page-compressed, and,
- Any of the in-row large object columns in the table are implicitly or explicitly LOB compressed.

CHAPTER 19 Configuring Shared Memory Dumps

Use the memory dump compression level configuration parameter to configure Adaptive Server to compress shared memory dump files.

Configuring Adaptive Server to use compressed shared memory dumps

Enabling memory dump compression level can significantly reduce the size of the shared memory dump files generated by Adaptive Server.

memory dump compression level

Summary information	
Default value	0
Valid values	1 – 9
Status	Dynamic
Display level	Comprehensive
Required role	System administrator
Configuration group	Diagnostics

memory dump compression level controls the compression level for shared memory dumps. The compression levels range from 0 (no compression) to 9 (highest compression). The speed of the compression is inversely proportional to the amount the dump is compressed. The lower the compression level, the faster Adaptive Server compresses the dump, but the size of the compressed file may be greater.

Configuring shared memory dumps

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 Adaptive Server processes the dump. See the *Reference Manual: Commands*.

Note Shared memory dump files are created to assist Sybase Customer Support in analyzing problems in Adaptive Server. Use sp_shmdumpconfig only under the direction of Sybase 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 Adaptive Server 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"
```