SYBASE®

An SAP® Company

New Features Guide

Adaptive Server® Enterprise

15.7 SP100

Contents

Change in Release Version Number

Software patches currently known to Sybase® customers as ESDs (Electronic Software Deliveries) following major or minor releases are now referred to as SPs (support packages), with numbers of up to three digits.

See SAP® Release Strategy for all Major Software Releases at: *https://service.sap.com/releasestrategy*. There is no change to upgrade or downgrade procedures because of this change in version number.

Shrinking Databases

As of Adaptive Server® version 15.7, you can shrink databases, freeing unused space for reuse or deletion.

If Adaptive Server 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 Adaptive Server in single user mode. See "Restrictions."

Note: The shrink database functionality is not supported on the Cluster Edition.

Shrinking a Database

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']
```

Shrinking a database on a device reduces the amount of free space for all segments on that device.

where:

- **off** – specifies the device names from which you are releasing space. The **off** clause includes:
  - *database_device* – is the name of the database device on which to locate the database extension. A database can occupy more than one database device with different amounts of space on each device.

    database_device includes either the *=size* parameter or the **from** or **to** parameters (not both). The default from page is 0 (zero). If you do not specify any modifiers with the **from** clause, Adaptive Server releases all storage for this database on the indicated device.
  - *size* – specifies the amount of space to be released from the database. **alter database** accepts a specific size, or a **from ... to** declaration:
    - Specific size – an integer. If you do not include a unit specifier, Adaptive Server uses megabytes as the unit. You can also include “k” or “K” (kilobytes), “m” or

"M" (megabytes), "g" or "G" (gigabytes), and "t" or "T" (terabytes). Sybase recommends that you always include a unit specifier. Quotes are optional if you do not include a unit specifier. However, you must use quotes if you include a unit specifier. If you do not provide a unit specifier, the value provided is presumed to be in megabytes. For example:

- **alter database ... off mydev1 = 200** – removes 200 megabytes
- **alter database ... off mydev1 = '2g'** – removes 2 gigabytes

- **from ... to** declaration – unsigned integers. See below.

Adaptive Server releases the highest-numbered logical pages of this database associated with the indicated device. Adaptive Server rounds the value for *size* up, if needed, to indicate an even number of allocation units.

If you specify more space than is currently in use by this database on the indicated device, Adaptive Server limits the specified size or page IDs to the amount of space available.

- **from** *page_number* – specifies the low end of the range of logical pages to be released. Adaptive Server rounds the **from** value down, if needed, to represent the first page of the allocation unit containing the indicated page. The default value for **from** is 0 (zero).
- **to** *page_number* – specifies the high end of the range of logical pages to be released. Adaptive Server rounds the **to** value up, if necessary, to represent the last page of the allocation unit containing the indicated page. The default value for **to** is the highest-numbered logical page in the database.

  However, if the **to** page is an allocation page itself (which is the starting page of an allocation unit) and is not the same as the **from** page, the **alter database ... off** command does not affect the allocation unit containing the specified **to** page. Adaptive Server assumes that a user requesting a precise page range does not intend to affect the specified end page, so it decrements the end page rather than increasing it.

Note: Use either **alter database off** or **alter database log off** to shrink the log segment. Both commands perform the same function.

- **with time=**'*time*' – specifies the maximum length of time **alter database ... off** may run, and is specified as hours, minutes, and seconds (*HH:MM:SS*). If you do not include the full specification, **alter database** interprets the missing sections for *time* as:
  - *XX:YY* is interpreted as *MM:SS*.
  - *XX* is interpreted as minutes.
  - Empty sections are treated as zero. For example, **alter database** interprets both "::30" and ":30" as 30 seconds. However, **alter database** interprets "1::" as 1 hour, and "2:" as 2 minutes.

  If **alter database ... off** does not finish within the specified period of time, work in progress is abandoned, and the command exits with an error message. Any completed work remains done.

Note: Adaptive Server does not issue an error message if you indicate a **from** page value that is greater than the **to** page value. Instead, Adaptive Server swaps the **from** and **to** values before applying the rounding logic.

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 Adaptive Server.

How Adaptive Server Shrinks the Database

For database shrink operations that do not include pages 0 – 255, Adaptive Server 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.

Adaptive Server 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 Adaptive Server 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.

Adaptive Server versions 15.7 SP100 and later have the text and image backlinking pointers created and maintained by default. For versions of Adaptive Server 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 Adaptive Server 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 Adaptive Server 15.7 SP100 does not require the above intensive operation. Adaptive Server 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.

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 Adaptive Server 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 Adaptive Server cannot allocate space within the correct segment to receive a page being moved. Any data already moved remains in its new location.

Adaptive Server 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

Adaptive Server 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 Adaptive Server to an earlier version. Earlier versions of Adaptive Server do not recognize suspect text indexes as being unusable, so Adaptive Server 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 Adaptive Server 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 Adaptive Server 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, Adaptive Server 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.

  Adaptive Server 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 Adaptive Server 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 Adaptive Server 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)**.

Enhancements to Backup and Restore

Adaptive Server 15.7 SP100 includes enhancements to the **dump** and **load** commands.

- The **dump database** command allows you to perform a cumulative backup, in which you make a copy of all the pages that have been modified in the database since the last full database dump.
- The ability to back up (**dump**) one database to restore (**load**) into another, using a dump history file.
- Integration of Tivoli Storage Manager with the dump history file.
- The **sybdumptran** utility, which allows you to generate a final transaction log dump from log devices from outside of Adaptive Server, when your server suffers a catastrophic failure.
- A new ability with **load database** for **listonly=create_sql** to include database options and attributes.

Cumulative Dump

The Adaptive Server **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 Adaptive Server 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 of Adaptive Server.

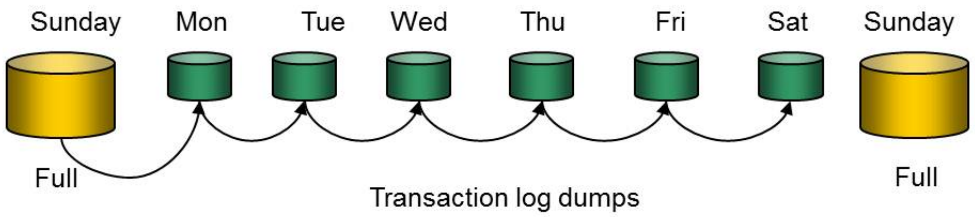
See also

- *dump database* on page 114

Dump and Load Sequences

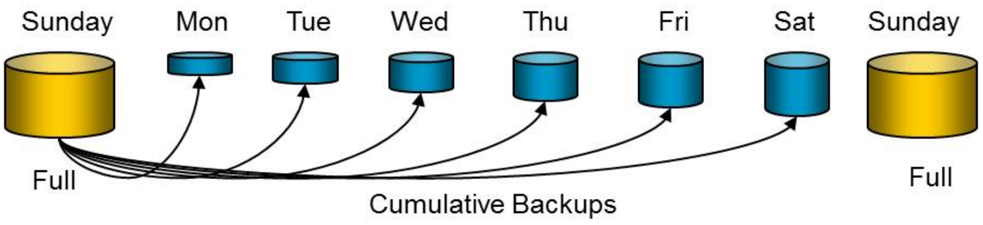
You can perform cumulative dumps instead of transaction log dumps between full database dumps.

In versions earlier than Adaptive Server 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

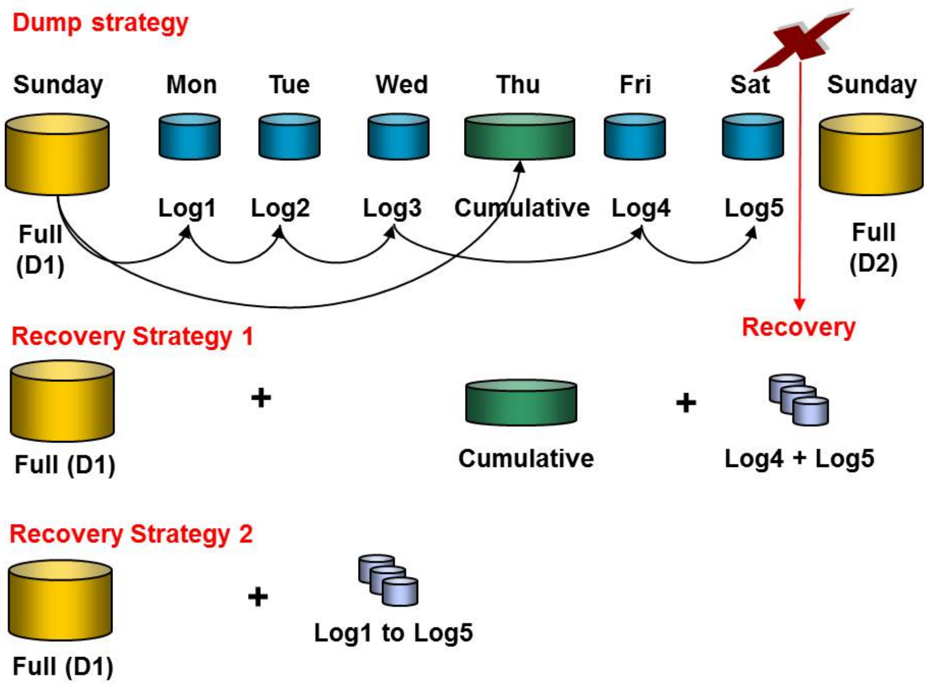
When you perform cumulative dumps, you need only load the most recent cumulative dump on top of its corresponding database dump.

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 1. 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 2. 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.
- (Adaptive Server 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 Adaptive Server earlier than 15.7 SP100.

System Changes Supporting Cumulative Dumps

Various commands and procedures support the cumulative dump feature in Adaptive Server 15.7 SP100.

Changes	Description
sp_dboption	Before you use the cumulative backup feature, enable a database to maintain the list of pages that need to be dumped. The **sp_dboption** includes this parameter, where *dbname* is the name of the database: `sp_dboption dbname, 'allow incremental dumps', true` When you include the `allow incremental dumps` parameter, Adaptive Server then automatically maintains the list of pages for the database, incurring a small performance overhead whether you actually perform a cumulative dump or not.

sp_configure	You can also configure Adaptive Server to enable concurrent transaction dumps and cumulative database dumps:
	`sp_configure 'enable concurrent dump tran', 1`
	You can execute concurrent **dump transaction** commands, but you cannot execute concurrent **dump database** (full) commands.
	
	You can record cumulative dump commands in the dump history file:
	`sp_configure 'enable dump history', 1`

Changes	Description

sp_dump_history	The **sp_dump_history** system procedure now includes the **cumulative** keyword in its *@dump_type* parameter, which supports all the functionality for cumulative dumps as it does for full database dumps. For example, run:
	`dump database mydb full to "/dev/db1.dmp"`
	`dump tran mydb to "/dev/dt1.dmp"`
	`dump tran mydb to "/dev/dt2.dmp"`
	`dump database mydb cumulative to "/dev/dc1.dmp"`
	`dump tran mydb to "/dev/dt3.dmp"`
	Then execute:
	`load database mydb with listonly=load_sql`
	The result is:
	`load database mydb full from "/dev/db1.dmp"`
	`load database mydb cumulative from "/dev/dc1.dmp"`
	`load tran mydb from "/dev/dt3.dmp"`
	Since the load sequence you get is the simplest one, if there are cumulative dumps, you need not use any transaction dump generated between the full database dump and the cumulative dump.

Changes	Description

sp_dump_info	The **sp_dump_info** system procedure estimates the size of data and log that a cumulative dump contains at a specific point in time. The size is reported in units of pages, KB, MB, or GB as appropriate, and may be slightly smaller than the actual size of the archive file (or files, if you are using multiple stripes), because the archive contains some additional information by way of the labels, header, trailer, and run list pages. By default, **sp_dump_info** assumes the dump is uncompressed. If it is compressed, the archive size is smaller than that reported by **sp_dump_info**. A sample output for **sp_dump_info**:
	`sp_dump_info test`
	`go`
	`Data    Log  Database percentage Allocation threshold`
	`------- ---- -------------------`
	`--------------------`
	`4368 KB 2 KB                  2                   40`
	`(return status = 0)  (return status = 0)`
	The output indicates that a cumulative dump occurring at this point in time, would contain approximately 4368KB of data and 2KB of log, representing 2 percent of the total database size.
	Compare this with the size of an actual cumulative dump of the archive file:
	`dump database test cumulative to "c:/tmp/test.dmp"`
	`go`
	`Backup Server: 4.171.1.1: The current value of 're-`
	`served`
	`pages threshold' is 85%.`
	`Backup Server: 4.171.1.2: The current value of 'allo-`
	`cated`
	`pages threshold' is 40%.`
	`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`
	`'test122480F0EF'`
	`section number 1 mounted on disk file 'c:/tmp/`
	`test.dmp'`
	`Backup Server: 4.188.1.1: Database test: 4328 kilo-`
	`bytes`
	`(3%) DUMPED.`
	`Backup Server: 3.43.1.1: Dump phase number 1 comple-`
	`ted.`
	`Backup Server: 3.43.1.1: Dump phase number 2 comple-`
	`ted.`
	`Backup Server: 3.43.1.1: Dump phase number 3 comple-`
	`ted.`
	`Backup Server: 4.188.1.1: Database test: 4370 kilo-`
	`bytes`
	`(3%) DUMPED.`

Changes	Description
	`Backup Server: 3.42.1.1: DUMP is complete (database test).` The corresponding size of the archive is 4487168 bytes, or 2191 pages. This differs from the estimate given by **sp_dump_info** by 29 pages (58KB), which is the result of 8 pages for the dump labels, 1 page for the dump header, 1 page for the dump trailer and 19 pages containing run lists. The size of the dump labels, header, and trailer are independent of the numbers of pages dumped, while the number of pages used by run lists is dependent on the numbers of pages dumped.
load database and load transaction	The **listonly=load_sql** option of **load database** and **load transaction** also takes into account any cumulative dumps stored in the dump history file.
loginfo	This builtin function displays information about the transaction log for a database.

See also

- *loginfo* on page 107
- *sp_configure* on page 123
- *sp_dboption* on page 125
- *sp_dump_history* on page 127
- *sp_dump_info* on page 121
- *dump database* on page 114
- *load database* on page 115

Changes to load database

Adaptive Server 15.7 SP100 includes changes to the **load database** command.

The **load database** and **load transaction** commands allow you to:

- Apply source database attributes to the target database using **load database** or (**load transaction**) with the **with listonly=create_sql** option.
- Enable **load database** and **load transaction** to specify a different target database using the **listonly=(create_sql, load_sql)** option.
- Integrate the dump history file with the Tivoli Storage Manager (TSM) being used as a repository for database and transaction log dumps.

Apply Source Database Attributes to the Target Database

With Adaptive Server 15.7 SP100, 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 Adaptive Server 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`, Adaptive Server 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

In Adaptive Server 15.7 SP100, 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)
```

Adaptive Server 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
```

Restore Databases in Tivoli Storage Manager to Different Target Databases

With Adaptive Server 15.7 SP100, you can load Tivoli Storage Manager-based backups into a different database on a different server when using the dump history file.

Earlier versions of Adaptive Server supported the use of **dump database** with TSM:

```
dump database source_dbname to "syb_tsm::object_name"
```

where:

- *source_dbname* – is the name of the source database.
- *syb_tsm* – is the fully qualified TSM name that specifies the name of the server and database from which you made the backup, using:

  ```
  -S source_server -D source_dbname
  ```
- *object_name* – is a name of the target database.

In Adaptive Server 15.7 SP100, the fully qualified TSM name is stored in the dump history file. Using the **load database... as** syntax allows you to restore a differently named database on a different server.

Performing this **dump database** command changes the information in the dump history file to correctly identify the fully qualified path of your object in TSM.

After you perform this command, the **sp_dump_history** system procedure displays the fully qualified TSM object name.

After you back up your database stored in TSM, you can restore it using:

```
load database source_dbname AS
      target_dbname with listonly=create_sql
```

The general form of the output generated is:

```
 load database target_dbname
            from syb_tsm::[[-S source_sever][-D source_dbname]
                  ::]object_name
            [stripe on syb_tsm::[[-S source_sever]
```

```
                  [-D source_dbname]::]object_name
           [[stripe on syb_tsm::[[-S source_sever]
                  [-D source_dbname]::]object_name
          load tran target_dbname
            from syb_tsm::[[-S source_sever]
                  [-D source_dbname]::]object_name
            [stripe on syb_tsm::[[-S source_sever]
                  [-D source_dbname]::]object_name
           [[stripe on syb_tsm::[[-S source_sever]
                  [-D source_dbname]::]object_name
```

The sybdumptran Utility

Use the **sybdumptran** utility to dump the most recent transactions when, due to the server suffering a catastrophic failure so that 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 Adaptive Server installation directory or a missing or corrupt master device, as **dump transaction with no_truncate** requires a working Adaptive Server 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 Adaptive Server 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 Adaptive Server catastrophic failure. **sybdumptran**, is unaffected by your inability to access parts of Adaptive Server, 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
```

See also

- *sybdumptran* on page 138

Aggregating Metrics from Syntactically Similar Queries

Adaptive Server 15.7 SP100 lets you 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 Adaptive Server 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.

Updates to Precomputed Result Sets

Adaptive Server version 15.7 SP100 includes updates to the commands and system procedures you use to manage precomputed result sets.

Updates to the precomputed result sets include:

- You can grant and revoke access permissions for precomputed result sets created with the **immediate refresh** parameter for the **select**, **update statistics**, and **delete statistics** commands.
- **sp_showoptstats** and **sp_depends** display additional information about precomputed result sets.

Improved Data Load Performance

Adaptive Server 15.7 SP100 improves data load performance by enabling **ins_by_bulk** optimization criteria. **ins_by_bulk** enables faster data loading by using bulk-data row inserts for **insert** statements.

When the data load optimization criteria is enabled with the **set** statement, Adaptive Server runs subsequent **insert...select** or batch insert statements with this optimization. The scope of the statement-level directive enabled via the abstract plan is limited to the statement.

This optimization improves data load performance on the target table with or without indexes.

See also

- *set* on page 117
- *Batch Insert Optimization* on page 37
- *Enable Data Load Optimization* on page 35

Enable Data Load Optimization

The optimization criteria **ins_by_bulk** improves data load performance.

Use **ins_by_bulk** at the session or query level.

- At the session level – use **set ins_by_bulk** to set the optimizer criteria for the current session. The syntax is:

  ```
  set ins_by_bulk {on | off}
  ```

- At the query level – set **ins_by_bulk** in the abstract plan for a specific **insert** statement.

  Note: The abstract plan name must be lowercase.

  The syntax is:

  ```
  insert into. . .
  select . . .
  plan "(use ins_by_bulk on)"
  ```

  For example:

  ```
  insert into my_salesdetail (stor_id, ord_num, title_id, qty,
  discount)
  select stor_id, ord_num, title_id, qty, discount from salesdetail
  where qty > 100
  plan '(use ins_by_bulk on)'
  ```

You can use either syntax (session level or query level) within a stored procedure.

See also

- *set* on page 117
- *select* on page 117
- *Batch Insert Optimization* on page 37
- *Improved Data Load Performance* on page 35

Tables with Indexes

Data load optimization with a target table that has indexes efficiently loads data pages while maintaining the index.

All rows are packed in a set of preallocated data pages and directly written to disk to improve data load performance. Recoverability is guaranteed.

When the target table defines indexes, the indexes are also maintained while writing data pages in bulk.

Note: **ins_by_bulk** is supported for tables with and without indexes.

Slightly increased database space may be required when the existing pages are not full because **insert...select** when used with **ins_by_bulk** does not append rows to the existing pages, but allocates a new page when inserting the first rows.

Viewing Queries with bulkInsertRowCount

show_cached_plan_in_xml displays the cached execution plan with **bulkInsertRowCount** to indicate the bulk insert mode. The number enclosed in the **bulkInsertRowCount** tag is the row count inserted under bulk insert mode.

This example shows that 10240 rows were inserted by bulk insert:

```
<Insert>
<VA>1</VA>
<est>
   <rowCnt>10240</rowCnt>
   <lio>9.749939</lio>
   <pio>9.749939</pio>
   <rowSz>2</rowSz>
</est>
<arity>1</arity>

<bulkInsertRowCount>10240</bulkInsertRowCount>

   <TableScan>
      ……
   </TableScan>
<objName>t1</objName>
<dataIOSizeInKB>16</dataIOSizeInKB>
```

```
<updateMode> Direct</updateMode>
</Insert>
```

show_cached_plan_in_xml is not available for **select into** because **select into** statements are not cached in Adaptive Server.

Batch Insert Optimization

Adaptive Server version 15.7 SP100 introduces batch insert optimization when the **insert** statement is dynamically prepared and executed as a batch via Open Database Connectivity (ODBC).

Batch insert optimization avoids extra back and forth messages from the client to Adaptive Server when **insert...values** statements uses dynamic parameters.

ins_by_bulk optimization further enhances batch insert performance by inserting rows in bulk mode internally.

The syntax is:

```
insert into table values (param_1, param_2, param_3)
```

Batch inserts also uses data-load optimization when **ins_by_bulk** is enabled.

For information about how to batch insert, see the *Adaptive Server Enterprise ODBC Driver 15.7 Users Guide*.

See also

- *set* on page 117
- *Improved Data Load Performance* on page 35
- *Enable Data Load Optimization* on page 35

Restrictions

Data load optimization is not followed when certain scenarios are met.

Data-Load Optimization – Restrictions

Traditional row-by-row mode data load is used when:

- **ins_by_bulk** is not enabled in the **plan** clause or **set** statement.
- The estimated page count to be inserted is fewer than eight pages (applies only to **insert...select** statements).
- The target table is a system table.
- The target table is a temporary table.
- The target table is a remote table.

- The target table is an all pages-locked (APL) table.
- The target table contains foreign keys for referential integrity.
- The target table is an in-memory database object.
- Referential integrity or deferred referential integrity is defined on the target table.
- **full** logging is not properly specified at the database, session, or query statement level.
- The target table contains large object (LOB) columns.
- The **insert** trigger is on the target table.
- **insert...values** is used without parameters.
- Source and target tables are the same.
- The **ignore_dup_key** option is used with a unique index.
- The batch insert query is not dynamically prepared.
- The insert operator is issued from the **alter table** command.
- Bulk insert is disabled while the following commands are underway concurrently when the data load is attempted: .
  - **reorg rebuild ... with online**
  - **create index ... with online**

  The insert instead performs in non-bulk mode, with serial index updates

Insert Bulk in a Transaction – Behavior and Restrictions

- Data loaded using bulk inserts in a multi-statement transaction is visible only to the inserting task while the transaction is active. While the transaction that loaded this data using bulk inserts is still active, concurrent access to this data by other tasks skips the uncommitted data.
- Concurrent data loads to the same table (or partition) from different tasks is supported. All data inserted during these concurrent data loads remains invisible to all tasks other than the inserting task until the inserting transaction commits.
- A single transaction can perform multiple bulk inserts to the same table multiple times. However, a maximum number of four tables per database can be inserted using this optimization in one transaction.

  For transactions that span multiple databases, each database has a maximum of four tables which is inserted using this optimization.

  Bulk inserts are not applied to the fifth and subsequent tables in one transaction, and data load is performed in non-bulk mode, with serial index updates.
- If the **ddl in tran** database option is enabled, an attempt to execute **create index ...with online** on a table following an insert bulk to that table in the same transaction is restricted. The **create index** operation is aborted, but the transaction remains active.

Incremental Reorganization

Adaptive Server 15.7 SP100 includes a new parameter—**defrag**—for the **reorg** command, which lets you schedule and resume reorganization, also allowing concurrent reads or writes on the data being reorganized.

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`.

Running reorg defrag

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.

Syntax:

```
reorg defrag table_name [partition {partition_list}]
   [with {time = hh:mm| resume | skip_compact_extents [= pct_value]}]
```

where:

- **defrag** – reorganizes each partition list or partition in the table while allowing concurrent reads or writes on the data being reorganized.
- **partition** – is the subset of the table that shares the same column definitions, constraints, and defaults as the table.
- *partition_list* – is the list of partition names.
- **time** – reorganizes the table or the list of partitions for the specified interval of time. *hh* is the number of hours and has no limit, and *mm* is the number of minutes 0–59.
- **resume** – resumes table reorganization from the end of the last reorganized data page invoked from the previous **reorg defrag**. **resume** continues until the entire table or list of partitions is reorganized, processing only those partitions that are currently either unvisited or partially reorganized. Running **time =** *hh:mm* with **resume** indicates that you are running reorganization from the previous position of reorganization and running it only for the specified interval of time.
- **skip_compact_extents** – skips compact extents. The compactness of an extent is measured as the percentage occupancy in that extent.
- *pct_value*– is the compactness of an extent measured as the percentage occupancy in that extent with a value of 1–100.

  Compactness = (Total space occupied in an extent / Total space in an extent) x 100

If **skip_compact_extents** is used, all the extents with compactness greater than or equal to the threshold occupancy percent value specified would be skipped for reorganization. If no threshold percent value is specified, the default percent value is 80%.

Examples:

In the following examples, 'partition list' is the list of data partitions specified in the command or if none specified, the list of all existing data partitions in the table.

- Example 1 reorganizes the partition list, starting from the beginning of each partition and continuing until its end:

  ```
  reorg defrag salesdetail [partition {seg1 [,seg2[, seg3]]}]
  ```
- Example 2 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  ```
  reorg defrag salesdetail [partition
     {seg1 [,seg2[,seg3]]}] with time = 01:20
  ```
- Example 3 restarts reorganization for each partition in the partition list that is either unvisited or partially reorganized, from where the previous invocation of reorganization has stopped, and if no point for resuming is available, reorganization starts from the beginning of the partition. Reorganization continues until the end of each partition.

  ```
  reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
     with resume
  ```
- Example 4 reorganizes the partition list, starting from the beginning of the partition and continuing until its end, skipping the extents with more than the specified occupancy threshold. If a percentage is not specified, extents exceeding 80% occupied are considered compact, and skipped.

  ```
  reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
     with skip_compact_extents [ = <1-100>]
  ```
- Example 5 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization stopped. If no point is available for resuming, reorganization starts from the beginning of the partition. Reorganization continues until the specified time interval is reached.

  ```
  reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
     with time = 01:20, resume
  ```
- Example 6 reorganizes the partition list, starting from the beginning of the partition and continuing until the specified interval of time completes, skipping each extent that qualifies to be compact as per the *pct_value* specified for **skip_compact_extents**. If a percentage is not specified, extents exceeding 80% occupied are considered compact, and skipped.

  ```
  reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
     with time = 01:20, skip_compact_extents [ = <1-100>]
  ```
- Example 7 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization stopped. If no point is available for resuming, reorganization starts from the beginning of the partition. Reorganization continues until the end of each partition. It skips extents that qualify to be compact as per the *pct_value* provided for **skip_compact_extents**. If the

pct_value is not provided, extents exceeding 80% occupied are considered compact and hence skipped.

```
reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
   with resume, skip_compact_extents[ = <1-100>]
```

- Example 8 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization has stopped. If no point is available for resuming, reorganization starts from the beginning of the partition. Reorganization continues until the specified time interval completes. It skips extents that qualify to be compact as per the *pct_value* specified for **skip_compact_extents**. If a percentage is not specified, extents exceeding 80% occupied are considered compact and hence skipped.

```
reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]
   with time = 01:20, resume, skip_compact_extents [ = <1-100>]
```

  For more information about **reorg**, see the *Reference Manual: Commands*.

For every 10% of a partition processed, this information appears:

```
Examined n allocation unit(s). Processed x pages out of y
data pages. z% completed, resulting in p% space compaction.
```

At the end of processing each partition, the time elapsed in the current invocation is printed as:

```
Elapsed time 1m : 56s : 623ms.
```

See also

- *reorg* on page 116

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" ] ]
```

See also

- *defrag_status* on page 105
- *sp_helpdefrag* on page 119

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.

The *System Changes* section in this guide provides syntax information.

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).

Restrictions

There are several restrictions that apply to **reorg defrag**.

- Tables must be data-only-locking (DOL) and have at least one index to use **reorg defrag**. Sybase recommends that the table has a unique index to efficiently accommodate the movement of the forwarded locations during reorganization.
- You can run only one instance of **reorg defrag** at a time on a table.
- When **reorg defrag** is in progress, DDLs cannot run on the table, for example add/drop partition, create/drop index, add/modify/drop columns. Also, other existing **reorg** subcommands like **rebuild**/**reclaim**/**compact**/**forwarded rows** cannot run when **reorg defrag** is in progress.

Creating Indexes Without Blocking Access to Data

Adaptive Server 15.7 SP100 includes the **create index ... online** parameter, which lets you create indexes without blocking access to the data you are indexing.

The syntax is:

```
create [unique] [clustered | nonclustered] index index_name
    on database.]owner.]table_name
    [with {...
            online,
    ...}
```

For example, to create the index `pub_dates_ix` on the `titles` table with the **online** parameter, use:

```
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
with online
```

Except for the **sorted_data** parameter, Adaptive Server processes other **create index** parameters the same way, both with or without the **online** parameter. For example, if you include the **reservepagegap** parameter with the **online** parameter, Adaptive Server reserves the pages while creating the new data layer. However, if you create the index using the **sorted_data** option, Adaptive Server creates the index on the existing data layer.

Restrictions

- User tables must include a unique index to use the **create clustered index ... online** command (creating nonclustered indexes does not have this restriction).
- You can run **create index ... online** with a **pll sort** only on round robin partitioned tables
- If you issue an **insert**, **delete**, **update**, or **select** command while **create index … online** or **reorg … online** are in the logical synchronization blocking phase:
  - The **insert**, **delete**, **update**, or **select** commands may wait and execute after **create index … online** or **reorg … online** are finished
  - Adaptive Server may issue error message 8233.
- You cannot:
  - Run **dbcc** commands and utility commands, such as **reog rebuild**, on the same table while you are simultaneously running **create index ... online**.
  - Run more than one iteration of **create index ... online** simultaneously.
  - Perform a **dump transaction** after running **create index ... online**. Instead, you can:
    - Run **create index ... online**, then dump the database, or
    - Run a blocking **create index**, then issue **dump transaction**.

- Run **create index ... online** within a multistatement transaction.
- Create a functional index using the **online** parameter.

Note: Because **create index ... online** increments the schema count in the `sysobjects` row that reflects the table's state change, concurrent activity waiting for **create index ... online** to commit may encounter error 540 after **create index ... online** commits.

Query Plan and Execution Statistics in HTML

In Adaptive Server versions 15.7 SP100 and later, you can generate a graphical query plan in HTML format for viewing in a Web browser.

In versions earlier than 15.7 SP100, analysis of complex queries evaluated text-based output provided by the **set showplan** and **set statistics {time | IO}** commands.

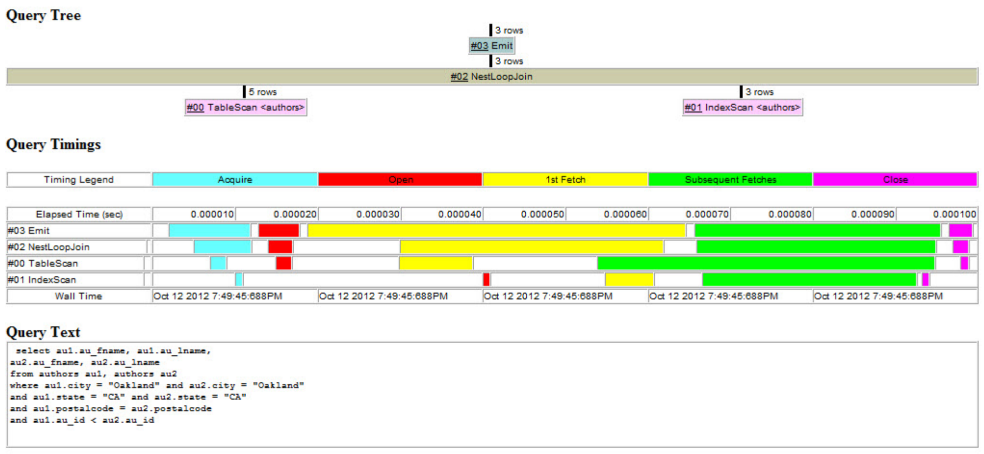
Use the HTML **set statistics** commands when a query plan is executed to generate graphical representations of query plan execution statistics. These HTML representations are on a single page, providing a comprehensive tree structure and timing diagram that offers a more intuitive and clearer description of a query plan.

Note: The query plan must be executed to retrieve the HTML representation for the plan.

The HTML output includes:

- The tree representation of the query plan provides information such as:
  - Number of row for each query operator and the connection between the parent and child operators.
  - Number of threads per operator if they query was executed in parallel by several threads.
- The full text of the query.
- Optionally, a timing diagram can be generated, which includes information such as:
  - A timing legend indicating the time spent in the different query execution phases, such as acquire, open, first fetch, subsequent fetches, and close.
  - Elapsed time for each operator and the execution phases for each query.
- For queries using parallelism, a thread-distribution diagram can be generated, which indicates the usage of threads along the query execution.
- A resource usage diagram can be generated, which displays CPU usage and wait-time distribution. CPU usage and Wait distribution is generated only for queries executed in parallel.

The following is an example of the HTML output, which includes a timing diagram.



```
 select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
and au1.au_id < au2.au_id
```

set statistics Command Options

The **set statistics** command allows generation of query plans in HTML format.

Use these **set statistics** options to generate query plans in HTML:

```
set statistics plan_html {on | off}
```

```
set statistics timing_html  {on | off}
```

```
set statistics plan_detail_html {on | off}
```

```
set statistics parallel_plan_detail_html {on | off}
```

```
set statistics plan_directory_html {dir_name | on | off}
```

See also

- *set* on page 117

Query Plan Tree Structure

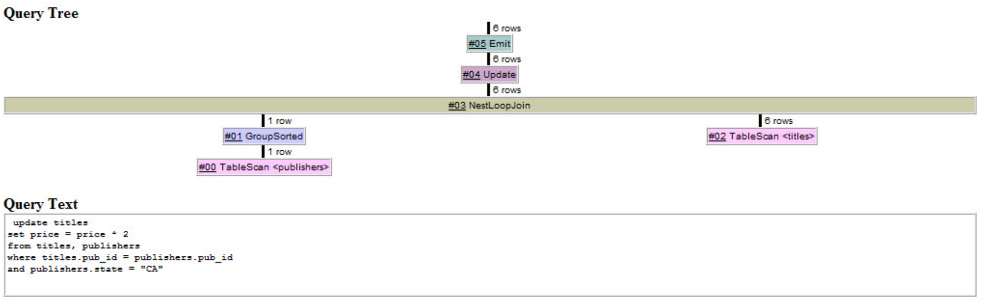
The query plan tree view shows the high-level query plan structure.

- In the tree view, each node represents a query operator, and the link between nodes shows the connection between parent and child operators.
- Operators of the same type use the same color, which indicates how data flows through the query plan nodes; from the leaf nodes to the top node of the plan.
- The links between operators are tagged with the actual number of rows exchanged at runtime between the child and parent node for that particular link.

- Hovering your cursor over the actual number of rows tag shows the number of estimated rows, which is calculated by the optimizer before the query is executed.
- The text that executes the query is included in the HTML output and can be used to perform analysis and comparison between the optimizer expected values and the real values at execution.

The following query uses the **set statistics plan_html** command to show the query plan tree in HTML format. The query is based on this example:

```
> set statistics plan_html on
> update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
> go
```

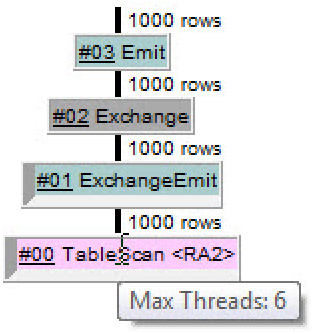


```
 update titles
set price = price * 2
from titles, publishers
where titles.pub_id = publishers.pub_id
and publishers.state = "CA"
```

The information provided for each operator includes:

- Operator name, such as Emit, Join, TableScan, Group By, and so on. Operator names are the same as those from the output of **show_plan_in_xml**, but might be different from the output of **set showplan on**.
- Operator internal ID number.
- Table name in the case of operators scanning or accessing tables.
- Index name for operators using indexes to access the tables.

For queries executed in parallel by multiple threads, hovering your cursor over the operator shows the number of threads involved in the execution of the operator. The degree of parallelism is shown by the size of the shadow box around that represents the operator: The larger the shadow, the higher the parallel degree.



In comparison, the same query is shown using the **set showplan on** command to view the query plan output in text format:

```
 > set showplan on
> update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
> go

STEP 1
  The type of query is UPDATE.

  5 operator(s) under root

  |ROOT:EMIT Operator (VA = 5)
  |
  |  |UPDATE Operator (VA = 4)
  |  |  The update mode is deferred_varcol.
  |  |
  |  |  |NESTED LOOP JOIN Operator (VA = 3) (Join Type: Inner Join)
  |  |  |
  |  |  |  |GROUP SORTED Operator (VA = 1)
  |  |  |  |Distinct
  |  |  |  |
  |  |  |  |   |SCAN Operator (VA = 0)
  |  |  |  |  |  FROM TABLE
  |  |  |  |  |  publishers
  |  |  |  |  |  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.
  |  |  |
  |  |  |  |SCAN Operator (VA = 2)
  |  |  |  |  FROM TABLE
  |  |  |  |  titles
  |  |  |  |  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.
  | |
  | |  TO TABLE
  | |  titles
  | |  Using I/O Size 2 Kbytes for data pages.
```

See also

- *set* on page 117

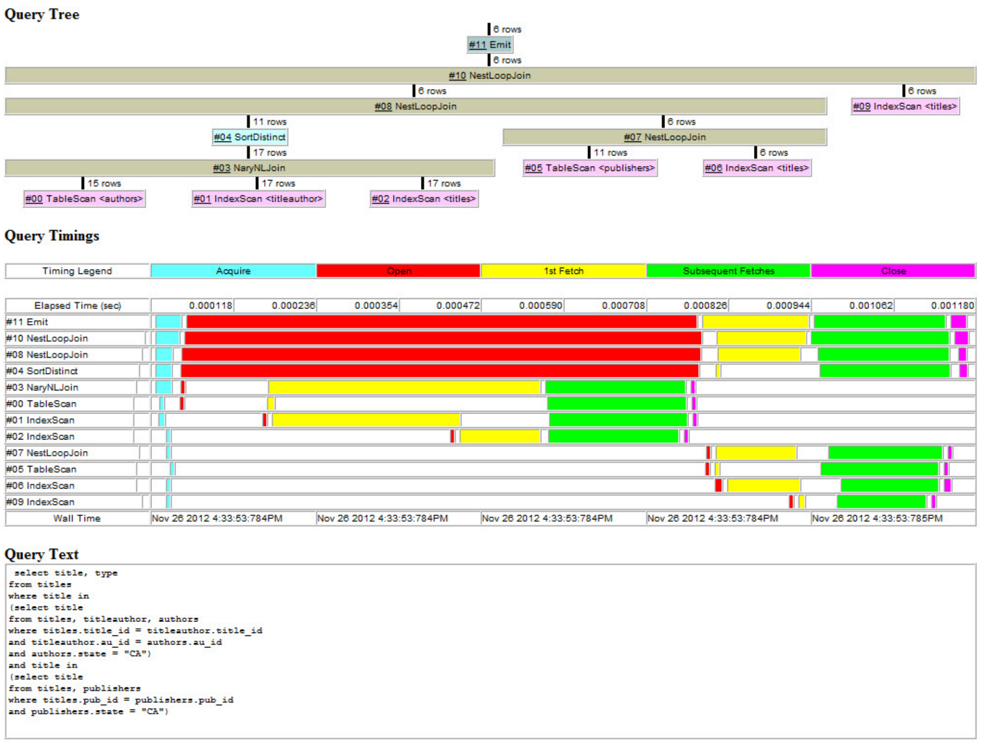
Query Timings

The length of time for each operator and the execution phases for each query are captured during the query execution.

To generate the timing diagram along with the HTML query plan, use **set statistics plan_html, timing_html on**.

The query that follows, executed to show the timing section of the HTML output, is based on this example:

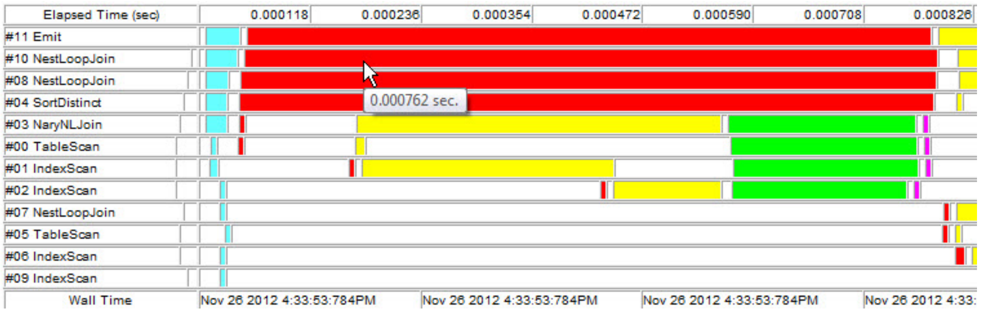
```
> set statistics plan_html, timing_html on
 > select title, type
  from titles
  where title in
    (select title from titles,
     titleauthor, authors
  where titles.title_id = titleauthor.title_id
    and titleauthor.au_id = authors.au_id
    and authors.state = "CA")
    and title in
   (select title from titles, publishers
  where titles.pub_id = publishers.pub_id
    and publishers.state = "CA")
> go
```



```
 select title, type
from titles
where title in
(select title
from titles, titleauthor, authors
where titles.title_id = titleauthor.title_id
and titleauthor.au_id = authors.au_id
and authors.state = "CA")
and title in
(select title
from titles, publishers
where titles.pub_id = publishers.pub_id
and publishers.state = "CA")
```

The Query Timings portion shows the statistics captured during the execution of the query. The distribution of the execution time is shown in two dimensions: per operator and execution phase. The timing legend indicates the time distribution for each operator in the different query execution phases such as acquire, open, first fetch, subsequent fetches, and close.

Hovering your cursor over the colored execution phase time bar displays the actual execution time. Execution time is expressed in seconds, with up to microsecond (6-digit) precision.



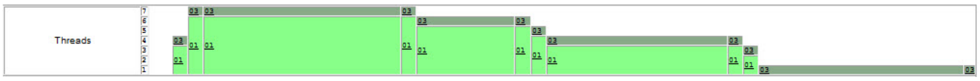
In comparison, the same query is shown using the **set statistics time** command to view the query plan output in text format:

```
> set statistics timing on
> select title, type
  from titles
  where title in
    (select title from titles,
     titleauthor, authors
  where titles.title_id = titleauthor.title_id
    and titleauthor.au_id = authors.au_id
    and authors.state = "CA")
    and title in
   (select title from titles, publishers
  where titles.pub_id = publishers.pub_id
    and publishers.state = "CA")
> go

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using Serial Mode

STEP 1
  The type of query is SELECT.

  11 operator(s) under root

   |ROOT:EMIT Operator (VA = 11)
   |
   | |NESTED LOOP JOIN Operator (VA = 10) (Join Type: Inner Join)
   | |
   | | |NESTED LOOP JOIN Operator (VA = 8) (Join Type: Left Semi Join)
   | | |
   | | | |SORT Operator (VA = 4)
   | | | | Using Worktable1 for internal storage.
   | | | |
   | | | | |N-ARY NESTED LOOP JOIN Operator (VA = 3) has 3 children.
   | | | | |
   | | | | | |SCAN Operator (VA = 0)
   | | | | | |  FROM TABLE
   | | | | | |  authors
   | | | | | |  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.
   | | | | |
   | | | | | |SCAN Operator (VA = 1)
   | | | | | |  FROM TABLE
   | | | | | |  titleauthor
   | | | | | |  Using Clustered Index.
   | | | | | |  Index : taind
   | | | | | |  Forward Scan.
   | | | | | |  Positioning by key.
   | | | | | |  Keys are:
   | | | | | |    au_id ASC
   | | | | | |  Using I/O Size 2 Kbytes for data pages.
   | | | | | |  With LRU Buffer Replacement Strategy for data pages.
   | | | | |
   | | | | | |SCAN Operator (VA = 2)
   | | | | | |  FROM TABLE
   | | | | | |  titles
   | | | | | |  Using Clustered Index.
   | | | | | |  Index : titleidind
   | | | | | |  Forward Scan.
   | | | | | |  Positioning by key.
   | | | | | |  Keys are:
   | | | | | |    title_id ASC
   | | | | | |  Using I/O Size 2 Kbytes for data pages.
   | | | | | |  With LRU Buffer Replacement Strategy for data pages.
   | | |
   | | | |NESTED LOOP JOIN Operator (VA = 7) (Join Type: Inner Join)
   | | | |
   | | | | |SCAN Operator (VA = 5)
   | | | | |  FROM TABLE
   | | | | |  publishers
   | | | | |  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.
   | | | | |
   | | | | |SCAN Operator (VA = 6)
   | | | | |  FROM TABLE
   | | | | |  titles
   | | | | |  Index : titleind
   | | | | |  Forward Scan.
   | | | | |  Positioning by key.
   | | | | |  Keys are:
   | | | | |    title ASC
   | | | | |  Using I/O Size 2 Kbytes for index leaf pages.
   | | | | |  With LRU Buffer Replacement Strategy for index leaf pages.
   | | | | |  Using I/O Size 2 Kbytes for data pages.
   | | | | |  With LRU Buffer Replacement Strategy for data pages.
   | |
   | | |SCAN Operator (VA = 9)
   | | |  FROM TABLE
   | | |  titles
   | | |  Index : titleind
   | | |  Forward Scan.
   | | |  Positioning by key.
   | | |  Keys are:
   | | |    title ASC
   | | |  Using I/O Size 2 Kbytes for index leaf pages.
   | | |  With LRU Buffer Replacement Strategy for index leaf pages.
   | | |  Using I/O Size 2 Kbytes for data pages.
   | | |  With LRU Buffer Replacement Strategy for data pages.
```

Thread Usage

For queries executed in parallel mode, the distribution of threads is shown as a bar diagram.

The graphical representation for threads shows the maximum number of active threads during the execution time of the query. The internal operator ID, which is included on the query tree diagram, shows the distribution and number of threads per operator.



In this diagram, the second column represents a particular section of the query execution time. The operator ID shows how many threads are being used. For example:

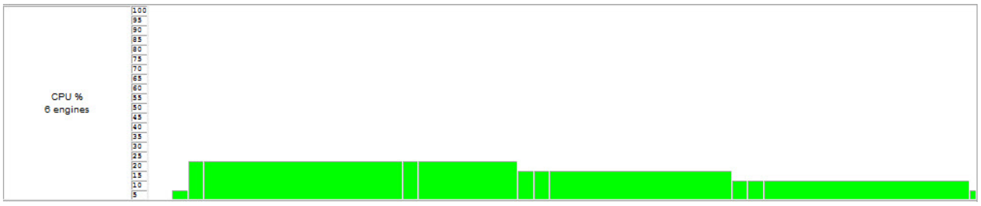
- Operator 01 is using six threads.
- Operator 03 is using one thread.

The column indicates that, at that particular time, the query was being executed in parallel by a maximum of 7 threads.

CPU Usage

For queries executed in parallel mode, the CPU usage by threads running a query is represented as a bar diagram.

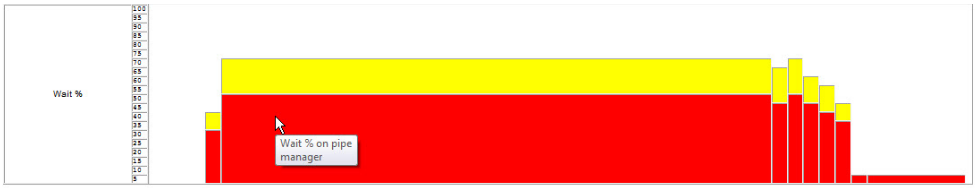
This diagram shows the percent of CPU used by threads running a query.

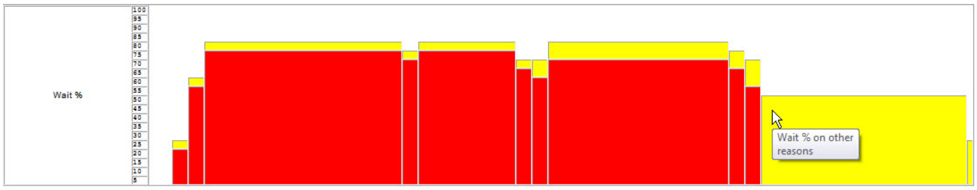


Wait Distribution

For queries executed in parallel mode, query time distribution for wait activities such as network I/O, disk I/O, or other wait conditions during the execution of the query are represented as a bar graph.

In this example, the red bars correspond to the amount of synchronization time spent for parallel execution structures, and the yellow bar represents other wait conditions, such as network I/O, disk I/O, and so on. The percentage indicates how much time is considered wait time in comparison with the CPU time that could have been used by all the threads active at that particular moment.

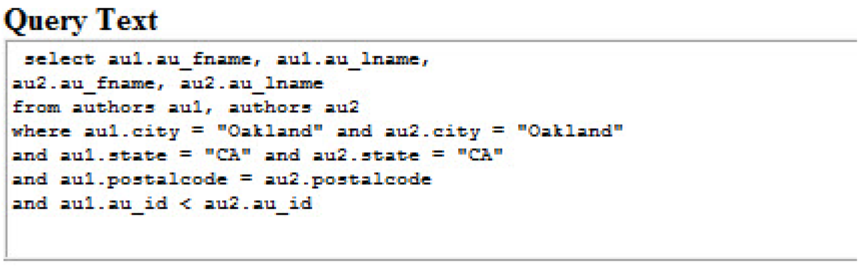




Query Text

The output of the executed query is provided in text format.

The text for the executed query is included with the HTML output for both query trees, and query trees with query timing. This is useful when this HTML output is archived in a file, allowing the printed text, which is the source of the query, to be kept with the rest of the diagnostic information.



Query Text

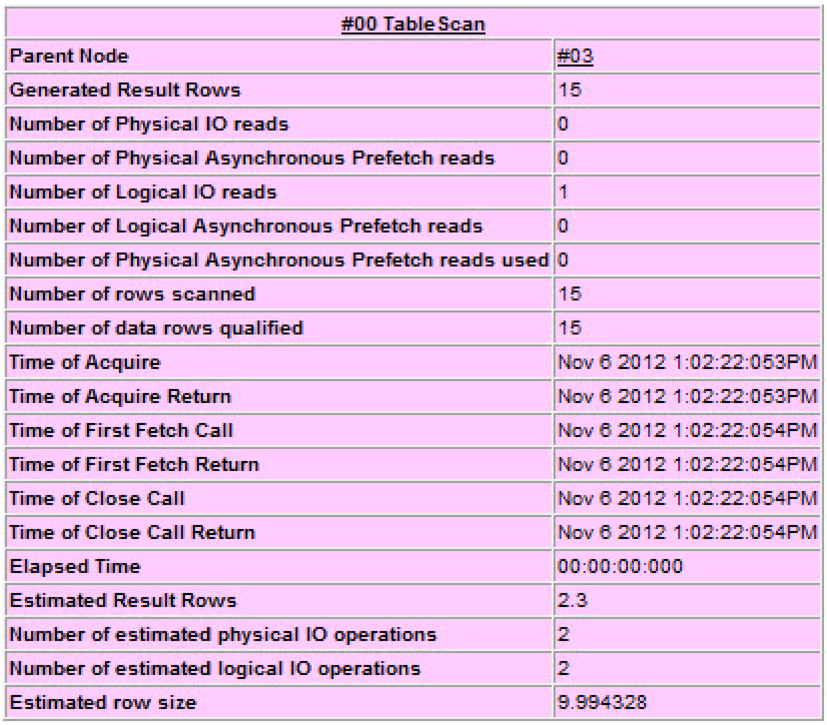
```
 select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
and au1.au_id < au2.au_id
```

Query Detail

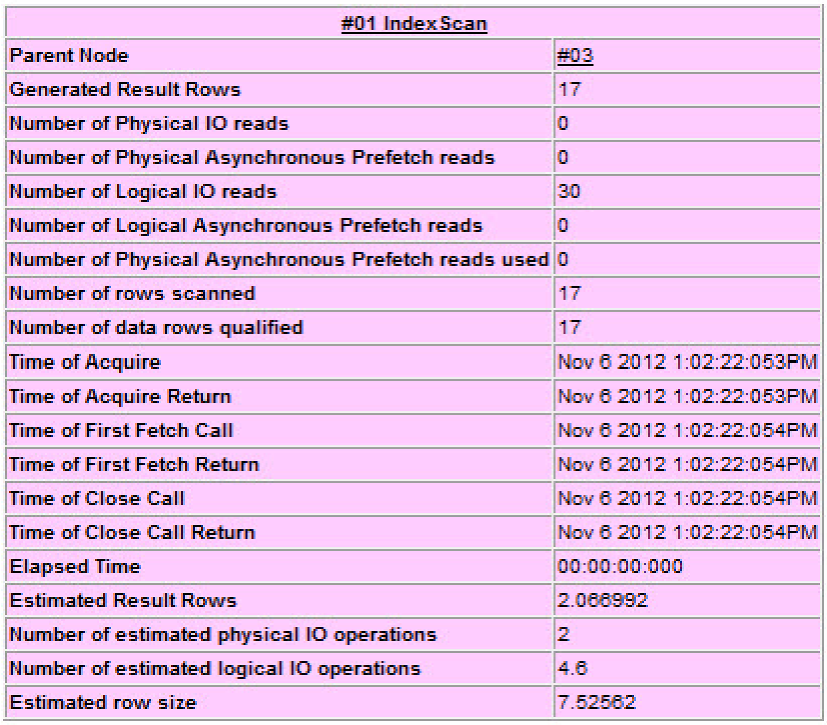
The detailed HTML generated for a query includes details for a specific operator.

To generate detailed information, use the **set statistics plan_detail_html** command. The detailed section includes information such as the name, different timestamps captured during the execution, number the rows affected, number of estimated rows, elapsed time, and so on.

The **set statistics parallel_plan_detail_html** command can be used to control the generation for worker threads plan fragments.



#00 Table Scan	
Parent Node	#03
Generated Result Rows	15
Number of Physical IO reads	0
Number of Physical Asynchronous Prefetch reads	0
Number of Logical IO reads	1
Number of Logical Asynchronous Prefetch reads	0
Number of Physical Asynchronous Prefetch reads used	0
Number of rows scanned	15
Number of data rows qualified	15
Time of Acquire	Nov 6 2012 1:02:22:053PM
Time of Acquire Return	Nov 6 2012 1:02:22:053PM
Time of First Fetch Call	Nov 6 2012 1:02:22:054PM
Time of First Fetch Return	Nov 6 2012 1:02:22:054PM
Time of Close Call	Nov 6 2012 1:02:22:054PM
Time of Close Call Return	Nov 6 2012 1:02:22:054PM
Elapsed Time	00:00:00:000
Estimated Result Rows	2.3
Number of estimated physical IO operations	2
Number of estimated logical IO operations	2
Estimated row size	9.994328



#01 Index Scan	
Parent Node	#03
Generated Result Rows	17
Number of Physical IO reads	0
Number of Physical Asynchronous Prefetch reads	0
Number of Logical IO reads	30
Number of Logical Asynchronous Prefetch reads	0
Number of Physical Asynchronous Prefetch reads used	0
Number of rows scanned	17
Number of data rows qualified	17
Time of Acquire	Nov 6 2012 1:02:22:053PM
Time of Acquire Return	Nov 6 2012 1:02:22:053PM
Time of First Fetch Call	Nov 6 2012 1:02:22:054PM
Time of First Fetch Return	Nov 6 2012 1:02:22:054PM
Time of Close Call	Nov 6 2012 1:02:22:054PM
Time of Close Call Return	Nov 6 2012 1:02:22:054PM
Elapsed Time	00:00:00:000
Estimated Result Rows	2.066992
Number of estimated physical IO operations	2
Number of estimated logical IO operations	4.6
Estimated row size	7.52562

See also

- *set* on page 117

Parallel Query Plan Fragments

Statistics information is generated at the thread level when the query is executed in parallel using several worked threads.

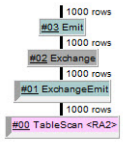
When the query is executed in parallel using several worked threads, several plan fragments (subplans from the main query plan) are created. Those plan fragments are executed in parallel by the threads.

Query tree and elapsed time per operator information is generated as HTML output at the thread level for each plan fragment and thread when **set statistics parallel_plan_detail_html** is on. The query tree shows only the fragment executed, and the timing diagram shows only the operators executed by that particular thread.

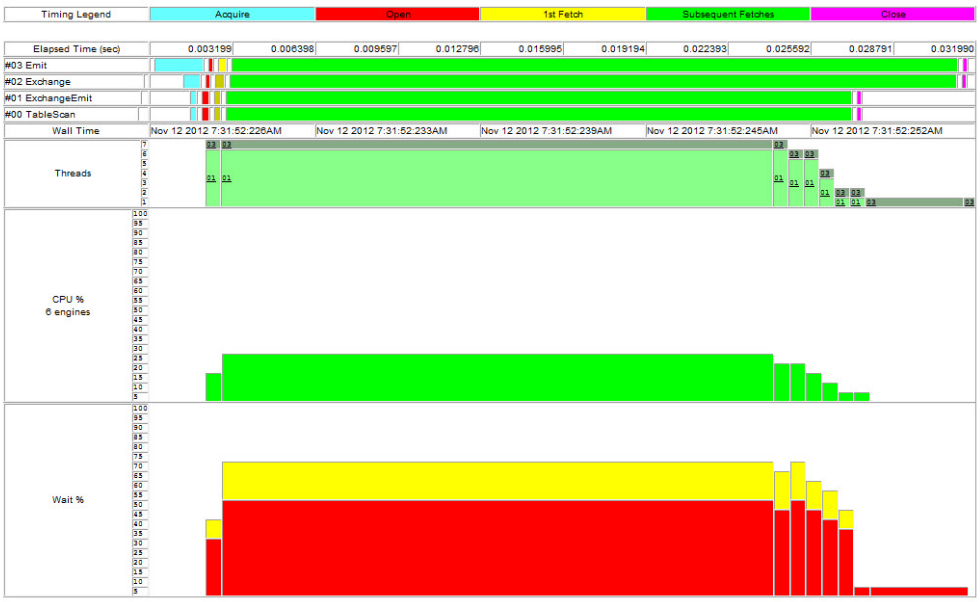
With this approach, you can diagnose the behavior of specific threads and plan fragments in comparison with the global execution of the query. This allows you to analyze the time spent in a particular operator by a specific thread, or the number of rows traversing an operator by thread, and so on.

For parallel queries, if you do not use the option **set statistics parallel_plan_detail_html**, only a single query tree and timing diagram is generated in HTML, representing the summary (in terms of execution time, rows and so on) for all the threads involved in the query.

Query Tree



Query Timings



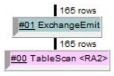
Query Text

```
select * from RA2 (parallel 6)
```

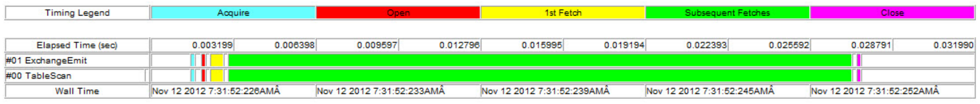
Threads

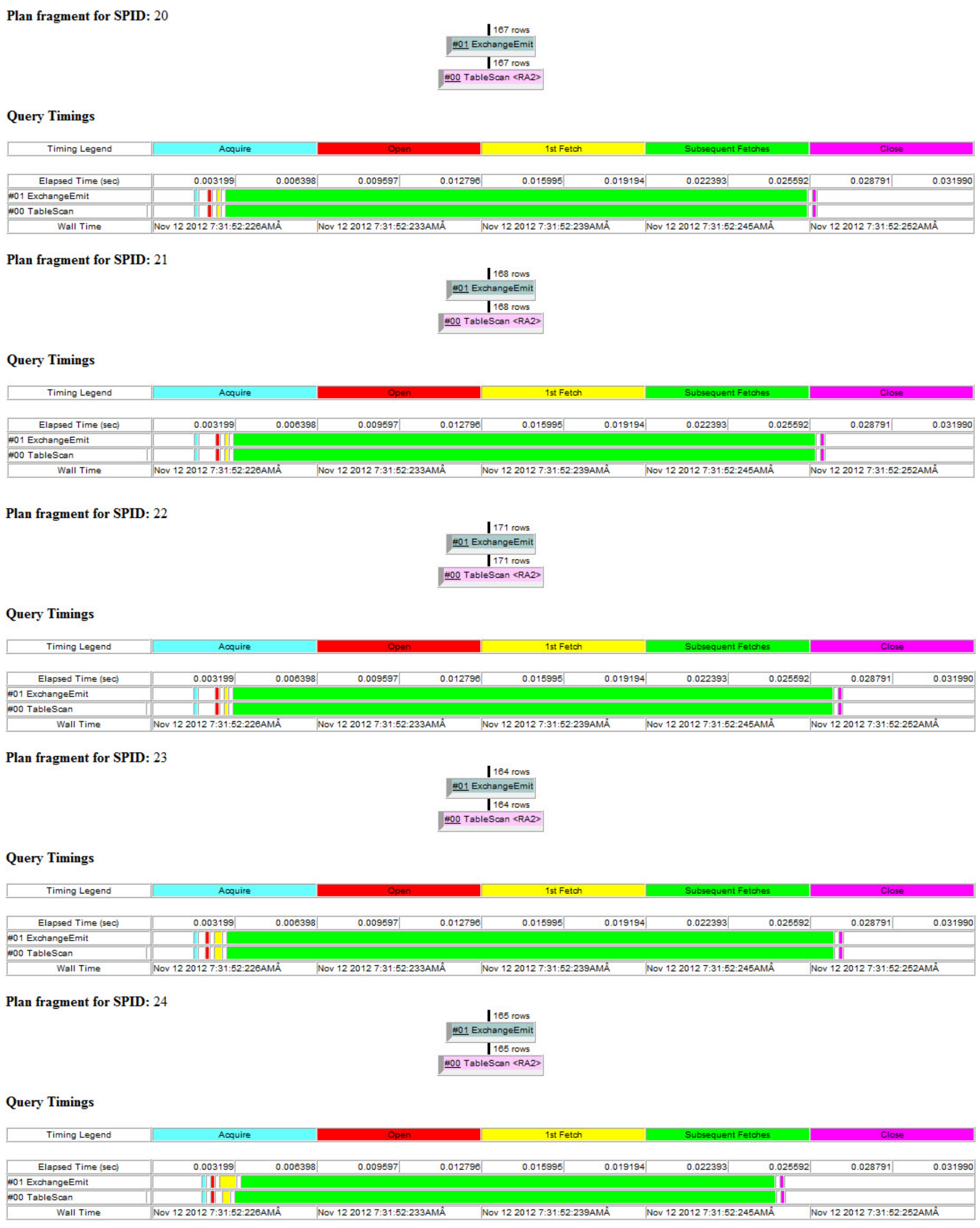
SPID	Thread ID	Rows produced
19	1376267	165
20	1245194	167
21	1114121	168
22	983048	171
23	851975	164
24	720902	165

Plan fragment for SPID: 19



Query Timings





Plan fragment for SPID: 20

167 rows
#01 ExchangeEmit
167 rows
#00 TableScan <RA2>

Query Timings

Timing Legend	Acquire	Open	1st Fetch	Subsequent Fetches	Close

Elapsed Time (sec)	0.003199	0.006398	0.009597	0.012796	0.015995	0.019194	0.022393	0.025592	0.028791	0.031990
#01 ExchangeEmit										
#00 TableScan										
Wall Time	Nov 12 2012 7:31:52:226AMÂ		Nov 12 2012 7:31:52:233AMÂ		Nov 12 2012 7:31:52:239AMÂ		Nov 12 2012 7:31:52:245AMÂ		Nov 12 2012 7:31:52:252AMÂ	

Plan fragment for SPID: 21

168 rows
#01 ExchangeEmit
168 rows
#00 TableScan <RA2>

Query Timings

Timing Legend	Acquire	Open	1st Fetch	Subsequent Fetches	Close

Elapsed Time (sec)	0.003199	0.006398	0.009597	0.012796	0.015995	0.019194	0.022393	0.025592	0.028791	0.031990
#01 ExchangeEmit										
#00 TableScan										
Wall Time	Nov 12 2012 7:31:52:226AMÂ		Nov 12 2012 7:31:52:233AMÂ		Nov 12 2012 7:31:52:239AMÂ		Nov 12 2012 7:31:52:245AMÂ		Nov 12 2012 7:31:52:252AMÂ	

Plan fragment for SPID: 22

171 rows
#01 ExchangeEmit
171 rows
#00 TableScan <RA2>

Query Timings

Timing Legend	Acquire	Open	1st Fetch	Subsequent Fetches	Close

Elapsed Time (sec)	0.003199	0.006398	0.009597	0.012796	0.015995	0.019194	0.022393	0.025592	0.028791	0.031990
#01 ExchangeEmit										
#00 TableScan										
Wall Time	Nov 12 2012 7:31:52:226AMÂ		Nov 12 2012 7:31:52:233AMÂ		Nov 12 2012 7:31:52:239AMÂ		Nov 12 2012 7:31:52:245AMÂ		Nov 12 2012 7:31:52:252AMÂ	

Plan fragment for SPID: 23

164 rows
#01 ExchangeEmit
164 rows
#00 TableScan <RA2>

Query Timings

Timing Legend	Acquire	Open	1st Fetch	Subsequent Fetches	Close

Elapsed Time (sec)	0.003199	0.006398	0.009597	0.012796	0.015995	0.019194	0.022393	0.025592	0.028791	0.031990
#01 ExchangeEmit										
#00 TableScan										
Wall Time	Nov 12 2012 7:31:52:226AMÂ		Nov 12 2012 7:31:52:233AMÂ		Nov 12 2012 7:31:52:239AMÂ		Nov 12 2012 7:31:52:245AMÂ		Nov 12 2012 7:31:52:252AMÂ	

Plan fragment for SPID: 24

165 rows
#01 ExchangeEmit
165 rows
#00 TableScan <RA2>

Query Timings

Timing Legend	Acquire	Open	1st Fetch	Subsequent Fetches	Close

Elapsed Time (sec)	0.003199	0.006398	0.009597	0.012796	0.015995	0.019194	0.022393	0.025592	0.028791	0.031990
#01 ExchangeEmit										
#00 TableScan										
Wall Time	Nov 12 2012 7:31:52:226AMÂ		Nov 12 2012 7:31:52:233AMÂ		Nov 12 2012 7:31:52:239AMÂ		Nov 12 2012 7:31:52:245AMÂ		Nov 12 2012 7:31:52:252AMÂ	

See also

- *set* on page 117

Writing HTML to an External File

You can write HTML generated query plan and execution statistics to a file in a specified directory.

The naming convention of the file is a combination of the user name, SPID, and the timestamp. Once a directory name is specified, you can turn the option off and then on again and resume writing to the named directory. If a directory name is not previously specified, HTML files are not generated.

This example turns on writing to an external file, and writes the HTML files to the directory `/usr/john/HTML`:

```
set statistics plan_directory_html "/usr/john/HTML"
go
set statistics plan_directory_html on
go
```

The directory provided must exist on the machine where the server is started. If you do not specify an absolute path, the relative path is taken from where the server was started.

See also

- *set* on page 117

Gathering Hash-Based Statistics with create index

Adaptive Server 15.7 SP100 extends the support of hash-based statistics gathering to the **create index** command.

The **create index** command hash-based statistics gathering options allow minor index attributes to be gathered while the index is being created serially or in parallel. With hash-based statistics gathering enabled, it is not necessary to update statistics after creating the index in regards to minor index attributes.

Use the following **create index** parameters to gather hash-based statistics. The syntax is:

```
create [unique] [clustered | nonclustered] index index_name
    on database.]owner.]table_name
    [with {...
            [, statistics {no_hashing | new_hashing | hashing}]

            [, statistics max_resource_granularity = int]
            [, statistics histogram_tuning_factor =int]
            [, statistics print_progress = int] } ]
    ...}
```

Enabling Hash-Based Statistics Gathering

Use **with statistics hashing** to indicate that an index is to be created, and minor index attributes are to be gathered using hash-based statistics. Use **with statistics no_hashing** to indicate that no hash-based statistics are to be gathered. Index-leading column statistics that are gathered using sort-based algorithm from versions of Adaptive Server versions earlier than 15.7 ESD #2 are not impacted by the **with statistics hashing** options.

sp_configure Option

A new **sp_configure** option, **utility statistics hashing** is introduced in Adaptive Server 15.7 SP100. For option settings, see **sp_configure** in the *System Changes > Procedures > Changed System Procedures* section.

If the clauses **no_hashing**, **new_hashing**, or **hashing** have not been specified in the **create index** command, the value of **utility statistics hashing** is used.

Precedence of Existing sp_configure Options

Any explicit phrases used in **create index with statistics histogram_tuning_factor** take precedence over the **sp_configure histogram_tuning_factor**.

Any explicit phrases used in **create index with statistics max_resource_granularity** take precedence over the **sp_configure max_resource_granularity**.

Examples

Example 1

Creates a unique clustered index named *au_id_ind* on the *au_id* and *title_id* columns of the *authors* table. Statistics are gathered with hashing, counts 50 steps, and enables **print_progress** which shows progress messages:

```
create unique clustered index au_id_ind
    on authors(au_id, title_id)
    with statistics hashing,
        statistics using 50 values,
        statistics print_progress = 1
```

Example 2

Creates an index named *ind1* on the *au_id* and *title_id* columns of the *titleauthor* table. Statistics are gathered with hashing, counts 50 steps, and sets the percentage of system resources a query can use at 80 percent using the **max_resource_granularity** option:

```
create index ind1
    on titleauthor (au_id, title_id)
    with statistics using 50 values,
        statistics hashing,
        statistics max_resource_granularity = 80
```

Example 3

Creates a nonclustered index named *zip_ind* on the *postalcode* and *au_id* columns of the *authors* table. Each index page is filled one-quarter full and the sort is limited to 4 consumer processes. Statistics are gathered with hashing, counts 50 steps, and generates an intermediate 40-step histogram:

```
create nonclustered index zip_ind
    on authors(postalcode, au_id)
    with fillfactor = 25,
        consumers = 4,
        statistics using 50 values,
        statistics hashing,
        statistics histogram_tuning_factor = 40
```

Note: You can also use **sp_configure** to set **max_resource_granularity** and **histogram_tuning_factor**. Hash-based statistics settings in the **create index** command take precedence over any values set by **sp_configure**.

Example 4

Creates a unique clustered index named *au_id_ind* on the *au_id* and *title_id* columns of the *authors* table. Statistics are gathered with hashing for minor attributed columns that have not had statistics previously gathered:

```
create unique clustered index au_id_ind
    on authors(au_id, title_id)
    with statistics new_hashing
```

See also

- *sp_configure* on page 123
- *create index* on page 111

Query Plan Optimization with Bloom Filters

Adaptive Server 15.7 SP100 introduces a query plan optimization feature, bloom filtering, which improves join performance.

A bloom filter provides early filtering of rows that cannot be joined before they would reach the join operator. Bloom filter is considered only when one child of the join operator must materialize its derived table before the other child is opened. The materializing child is called the build side. The bloom filter uses a bit array to represent qualifying join key values from the build side. The join then pushes the bit array to the other side, called the probe side. The probe side then uses the bit array to test whether a row, with certain join key values, might match the build side values. If not, the row is filtered from further processing. Bloom filter provides a low-cost approach for eliminating rows before they participate in a full join algorithm.

In Adaptive Server, bloom filtering is implemented for hash joins, sort merge joins, and reformatting based nested loop joins. It is applicable only to equi-joins. For hash joins, the build side is part of the hash join itself. For sort merge joins, the build side is the left side sort operator. For reformatting based nested loop joins, the build side is the insert operator for the worktable index creation. Bloom filtering has an added CPU cost when it builds and probes the bit array. The optimizer uses bloom filtering in the final plan if doing so is estimated to reduce the total cost of the join.

The bloom filter is enabled only when optimization criteria **join_bloom_filter** is on. The **join_bloom_filter** is on by default under the **allrows_dss** optimization goal when the optimization level is set to match the 15.7 SP100 or later, as in **ase_current**.

To view all available current optimizer settings, enter:

```
sp_options "show"
```

The memory cost of the bit array has an upper bound of about 512K bytes for the bit array of each bloom filter during execution. For most scenarios, memory cost less than 16K.

The performance of bloom filtering mostly benefits large joins. Test results for query execution elapse time in Adaptive Server with bloom filters have shown reductions of 15% to 60%. The added overhead to the plan execution is often very small. The benefits also vary depending on the datatypes of the join key columns. Integer datatypes show the best potential performance gain, while unichar and univarchar datatypes show less.

See also

- *set* on page 117

Example of showplan Output for Bloom Filters

The output of **showplan** indicates the use of bloom filters in the final plan.

Any use of bloom filters is indicated in **showplan** output. A single plan can contain multiple bloom filters. Each bloom filter is associated with a unique ID, represented in the output as "bv<id>".

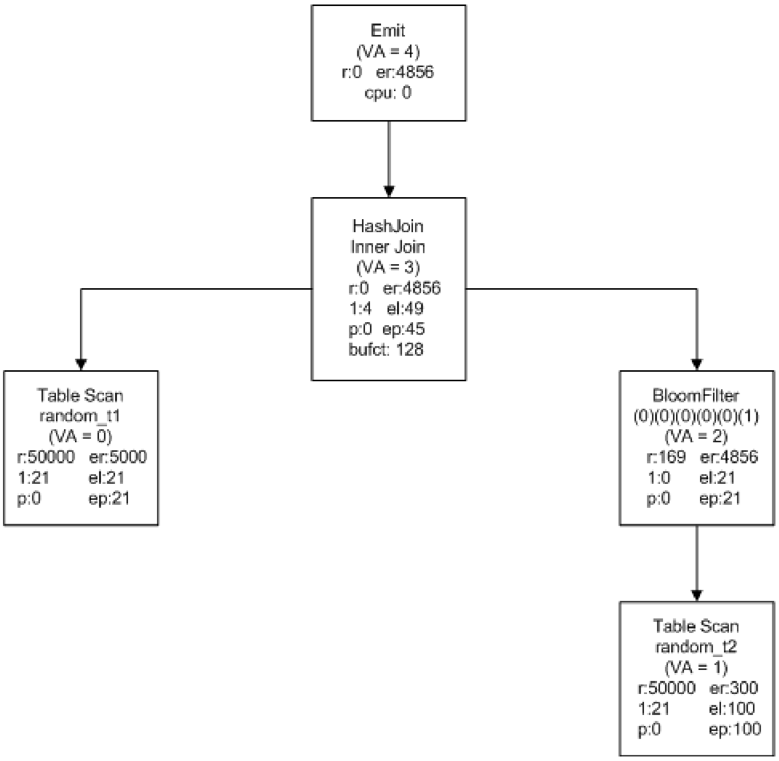
An example of **showplan** output:

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using Parallel Mode
Optimized using the Abstract Plan in the PLAN clause.
STEP 1
The type of query is SELECT.
8 operator(s) under root
| ROOT:EMIT Operator (VA = 8)
|
| | HASH VECTOR AGGREGATE Operator (VA = 7)
| | GROUP BY
| | Evaluate Grouped COUNT AGGREGATE.
| | Evaluate Grouped SUM OR AVERAGE AGGREGATE.
| | Using Worktable4 for internal storage.
| | Key Count: 1
| |
| | | HASH JOIN Operator (VA = 6) (Join Type: Inner Join)
| | | Using Worktable3 for internal storage.
| | | Building pushdown bloom filter (bv6).
| | |  Key Count: 1
| | |
| | | | SCAN Operator (VA = 0)
| | | | FROM TABLE
| | | | Location
| | | | L
| | | | Table Scan.
| | | | Forward Scan.
| | | | Positioning at start of table.
| | | | Using I/O Size 16 Kbytes for data pages.
| | | | With LRU Buffer Replacement Strategy for data pages.
| | |
| | | | HASH JOIN Operator (VA = 5) (Join Type: Inner Join)
| | | | Using Worktable2 for internal storage.
| | | | Building pushdown bloom filter (bv5).
| | | | Key Count: 1
| | | |
| | | | | SCAN Operator (VA = 1)
| | | | | FROM TABLE
| | | | | Customer
| | | | | C
| | | | | Table Scan.
| | | | | Forward Scan.
| | | | | Positioning at start of table.
| | | | | Using I/O Size 16 Kbytes for data pages.
| | | | | With LRU Buffer Replacement Strategy for data pages.

| | | |
| | | | | HASH JOIN Operator (VA = 4) (Join Type: Inner Join)
| | | | | Using Worktable1 for internal storage.
| | | | | Building pushdown bloom filter(bv4).
| | | | | Key Count: 1
| | | | |
| | | | | | SCAN Operator (VA = 2)
| | | | | | FROM TABLE
| | | | | | Time
| | | | | | T
| | | | | | Table Scan.
| | | | | | Forward Scan.
| | | | | | Positioning at start of table.
| | | | | | Using I/O Size 16 Kbytes for data pages.
| | | | | | With LRU Buffer Replacement Strategy for data pages.

| | | | |           |
| | | | | RESTRICT Operator (VA = 4)
| | | | | | Using pushdown bloom filter (bv4, bv5, bv6).
| | | | | |
| | | | | | | SCAN Operator (VA = 3)
| | | | | | | FROM TABLE
| | | | | | | Orders
| | | | | | | O
| | | | | | | Table Scan.
| | | | | | | Forward Scan.
| | | | | | | Positioning at start of table.
| | | | | | | Using I/O Size 16 Kbytes for data pages.
| | | | | | | With MRU Buffer Replacement Strategy for data pages.
```

Example of Lava Operator Tree for Bloom Filters

The lava operator tree indicates the use of bloom filters.

When using the **set statistics plancost on** command, the bloom filters operator is listed in the lava operator tree. By comparing the actual row count of the child operator and the actual row count of the bloom filter operator, you can calculate how many rows have been filtered out by bloom filter probing.



Disabling Bloom Filters

To disable bloom filters, use the **set join_bloom_filter** command.

For example, to disable bloom filters at the session level, enter:

```
1> set join_bloom_filter off
2> go
```

Adaptive Server Support for Multiple Scanner Threads

The Adaptive Server monitoring tables support multiple scanner threads for RepAgent. Enabling multiple scanner threads for RepAgent provides access to available replication paths and improved replication performance.

Adaptive Server monitoring tables

When you enable multiple scanners, the Adaptive Server monitoring tables provide information about RepAgent scanner task performance. These monitoring table changes have been introduced:

- `monRepSenders` – these columns have been added to provide compatibility with the single-task model.
  - `NumberOfTruncPointRequested`
  - `NumberOfTruncPointMoved`
  - `AvgTruncPointInterval`
- `monRepScanners` –

  the `NumberOfTruncPointRequested` and `NumberOfTruncPointMoved` columns are populated when the single-task scanning of a log model is used. When the Multi-Path Replication (MPR) model is used, these columns values are 0. The related information for the MPR model can be found on the `monRepSenders` table.
- The `monRepCoordinator` table has been added.

Any MPR binding model can be used with multiple scanners. For more information about multiple scanner threads, see:

Multiple Scanners in the *Replication Server New Features Guide*.

See also

- *Changed Monitoring Tables* on page 135

Adaptive Server Support for Replication by Column Value

Adaptive Server 15.7 SP100 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 Adaptive Server 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.

Adaptive Server System Table Storage for Replication Filters

A bound replication filter object is stored in these Adaptive Server 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 Adaptive Server 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*.

See also

- *sp_config_rep_agent* on page 124
- *sp_replication_path* on page 129
- *Changed Monitoring Tables* on page 135
- *monRepSchemaCache* on page 133

Sybase Central Adaptive Server Plug-in

Sybase no longer maintains the Sybase Central Adaptive Server plug-in.

For Adaptive Server versions 15.7 SP100 and later, use Sybase Control Center for Adaptive Server. Sybase Control Center provides a single, comprehensive, Web-administration console for real-time performance, status, and availability monitoring of large-scale Sybase enterprise servers. For more information, see the Sybase Control Center product documentation at *http://sybooks.sybase.com/*.

Running Adaptive Server on VMware vSphere 5.0

Adaptive Server version 15.7 SP100 adds support for VMware vSphere 5.0.

After you install Adaptive Server on VMware, Sybase recommends that you:

- Examine the BIOS settings, disabling any unnecessary processes and peripherals
- Direct critical computing resources (CPU, memory, network, and I/O) to the databases
- Install the VMware tools on the guest operating system

Note: Refer to the VMware Compatibility Guide to view hardware supported by VMware. Sybase's internal testing determined Adaptive Server's performance may depend on the vSphere configurations for the hardware you chose. Refer to VMware published best practices for running vSphere on the hardware.

Configure the BIOS Settings

Sybase recommends that you change some BIOS settings before you run Adaptive Server on the VMware guest operating system.

Enable:	Disable:
• Virtualization technology • Turbo mode • Hardware-based virtualization support • Hyperthreading (workload dependent) • Wake on LAN (required for VMware vSphere Distributed Power Management) • Execute Disable (required for vMotion and DRS) • Static high performance and balanced policies	• Node interleaving • C1E halt state • Power saving • Any unused features, such as: • Video BIOS shadowable • Video RAM cacheable • On-board audio • On-board modem • On-board serial ports • On-board parallel ports • On-board game port • Floppy disk drive • CD-ROM drive • USB port

See your operating system documentation.

Configure the Guest Operating System

After the installation is complete, disable unnecessary foreground and background processes.

Note: Sybase strongly recommends that you install the VMware Tools on the guest operating system. VMware Tools includes utilities that enhance the performance of the virtual machine's guest operating system and improve management of the virtual machine.

Check the Adaptive Server release bulletin for your platform for additional operating system and version settings and dependencies.

CPU Considerations

To improve performance for latency-sensitive applications running on guest operating systems, schedule all virtual machines on the same Non-Uniform Memory Access (NUMA) node. Allocate all virtual memory from the local physical memory attached to that NUMA node.

When you schedule processor affinity for vCPUs on a specific NUMA node, use the vSphere Client to set the processor affinity:

- To change the VM settings for processor affinity:
  1. In the vSphere Client, select **Resources tab** > **Advanced CPU**.
  2. Select **Scheduling Affinity**.
- To change the virtual machine settings for memory affinity:
  1. In the vSphere Client, select **Resources tab** > **Advanced Memory**.
  2. Select **NUMA Memory Affinity**.

Recommendations for Virtual CPUs

In a production environment, verify that the total virtual CPU resources for the virtual machines do not exceed the CPU capacity of the host machine. That is, the total number of CPUs for the virtual machine cannot exceed the CPU capacity of the physical machine.

Sybase recommends that you undercommit CPU resources on the host machine; if the host CPU capacity is overloaded, the virtual database performance may degrade. Never overcommit CPU resources in a production environment.

A reasonable upper limit for CPU resources is about 80 percent of consumption. Sybase recommends that you create an alert to be sent to the virtual infrastructure administrator when CPU resources reach 90 percent of consumption.

Multithreaded applications like Adaptive Server may benefit from using multiple CPUs. To ensure that the physical CPUs are not oversubscribed, limit the total number of virtual CPUs to the total number of physical CPUs minus 1.

Configuring virtual Adaptive Servers with excess virtual CPUs can impose a small resource requirement on vSphere because unused virtual CPUs continue to consume timer interrupts. vSphere coschedules virtual CPUs and attempts to run the virtual CPUs in parallel whenever possible. Unused virtual CPUs impose scheduling constraints on the used virtual CPUs and might degrade performance.

CPU Scheduling

You may see improved performance if you ensure that the virtual machine remains scheduled when the vCPU is idle may improve performance.

To determine whether a scheduled virtual machine is providing optimal database performance:

- Set **monitor_control.halt_desched** to false.
- vCenter (the VMware management utility) should normally monitor performance. However, periodically use the vSphere Client to collect additional statistical measurements, or use **esxtop** or **resxtop**.

  esxtop provides performance insight concerning CPU resource contention. Database administrators should work with their VMware administrator to interpret **esxtop** statistics. Generally:

  - The system is overloaded if the load average listed on the first line of the **esxtop** CPU Panel is greater than or equal to the number of physical processors on the system.
  - The "usage percentage of physical CPUs" on the PCPU line may indicate an overloaded system.

    A reasonable upper limit in production environments is 80% usage; 90% usage should be used as an alert to the VMware administrator that the CPUs are approaching an overloaded condition. However, make decisions concerning usage levels based on the criticality of the database you are virtualizing and the desired load percentage.
  - These statistics are critical:
    - `%RUN` – percentage of total time the processes on the VMkernel are running on the processor; a high value for `%RUN` does not necessarily indicate that the virtual machine is resource constrained.
    - `%RDY` – percentage of time the processes on the VMkernel are ready to run, but are not scheduled to a core. A process on the VMKernel in a run queue is waiting for the CPU scheduler to let it run on a PCPU. A value of greater than 10% may indicate resource contention.
    - `%CSTP` – percentage of time the processes on the VMkernel are stopped to allow other virtual CPUs on the virtual machine to catch up (this is the unscheduled state). A value greater than 5% typically indicates that the virtual machine's workload is not balanced.

Memory Considerations

Configure the value for virtual memory equal to the value of the Adaptive Server **max memory** configuration parameter.

When consolidating Adaptive Server instances, vSphere allows you to share memory across virtual machines that may be running the same operating systems, applications, or components. In this situation, vSphere uses a proprietary transparent page-sharing technique to reclaim memory, and allows databases to run with less virtual memory than physical memory (see your VSphere documentation). You can use transparent page sharing for consolidation purposes in a test or development environment. However, never overcommit on production environments.

Resource Management

vSphere advanced workload management features, such as VMware vMotion and VMware DRS, free Adaptive Server from the resource limitations of a single host.

You can use vMotion to move running Adaptive Server virtual machines from one physical vSphere to another to balance available resources. DRS can dynamically allocate and balance computing resources by continuously monitoring the resource pools associated with virtual machines in a VMware cluster.

You can safely overcommit virtual resources for Adaptive Server in nonproduction environments where predictable and consistent performance is not critical.

Hardware-Assisted Memory Virtualization

Some processors address memory management unit (MMU) virtualization overhead by providing hardware support that virtualizes the MMU.

Without hardware-assisted MMU virtualization, VMware ESX maintains “shadow page tables” that directly map a guest's virtual memory to a host's physical memory address.

These shadow page tables are maintained for the processor, and are kept synchronized with the guest page tables. This allows ordinary memory references to execute without additional overhead, since the hardware translation lookaside buffer (TLB) caches direct guest virtual memory to a host's physical memory address translations that are read from shadow page tables. However, extra work is required to maintain the shadow page tables.

Hardware assistance eliminates software memory virtualization overhead. In particular, it eliminates the overhead required to keep shadow page tables synchronized with guest page tables, although the TLB miss latency is significantly higher than without hardware assistance. This means that hardware assistance provides workload benefits that depend primarily on the memory virtualization overhead that is caused when using software memory virtualization.

If a workload involves a small amount of page table activity (for example, process creation, mapping the memory, or context switches), software virtualization does not cause significant overhead. However, workloads that include a large amount of page table activity, such as workloads from a database, are likely to benefit from hardware assistance.

Sybase recommends that you allow vSphere to choose the virtual machine monitor based on the combination of CPU and guest operating system. In most cases, you see better performance by using hardware-assisted virtualization rather than by using shadow page tables.

Linux Huge Pages

Linux huge pages can increase the efficiency of translation lookaside buffer access, improving database performance.

Using huge pages at the operating system level allows for approximately a 10% improvement in Adaptive Server performance; Sybase recommends their use in any 64-bit production system. In larger memory configurations (that is, configurations where Adaptive Server uses more than 256GB RAM), huge pages are required to start Adaptive Server.

Adaptive Server versions 15.0.3 and later support huge pages. By default, huge page support is enabled in VMware ESX versions 3.5 and later.

Networking

Sybase recommends that you use VMXNET 3 virtual network interface cards (NICs) for your Adaptive Server running on Windows virtual machines.

During internal testing, Sybase noted performance improvement for sites using the VMXNET3 NICs running on the Windows guest operating system compared to sites using the default E1000 adapter.

VMXNET3 supports an adaptive interrupt coalescing algorithm, which helps drive high throughputs to virtual machines using multiple vCPUs with parallelized workloads (for example, multiple threads).

Storage

vSphere uses datastores to store virtual disks. Datastores provide an abstraction of the storage layer that hides the physical attributes of the storage devices from the virtual machines.

VMware administrators can create a single datastore to use as a consolidated pool of storage, or they can create multiple datastores to isolate various application workloads.

If you use a traditional storage area network (SAN) deployment, Sybase recommends that you create a dedicated datastore, which allows database administrators to define individual service level guarantees for different applications, similar to provisioning dedicated logical unit numbers (LUNs) for physical disks. You can use vSphere features such as VMware vSphere Storage I/O Control, Storage vMotion, and Storage DRS to prioritize and manage datastore workloads.

Sybase recommends that you use dedicated datastores for production Adaptive Servers, and consolidated datastores for less demanding test or development environments.

Automatic Physical Database Rearrangement on Load

When loading a dump of a database that had a segregated log and data segment, Adaptive Server 15.7 SP100 and later will now rearrange the physical layout of the target database to ensure physical separation of the log and data segments.

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. Adaptive Server 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 Adaptive Server 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
```

Adaptive Server 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.

How load Handles Fragments in Adaptive Server 15.7 SP100 and Later

To avoid data and log fragments on the same device, Adaptive Server now rearranges the physical layout of the database to accommodate for the log and data distribution in the database dump. These Adaptive Server changes have been made:

- Adaptive Server 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, Adaptive Server only checked that the total size of the target database was large enough to accommodate the database dump of the source. Adaptive Server 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 as was used earlier, the following example shows the results when there is not enough 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 Adaptive Server now handles loading a database during which physical rearrangement has occurred. Adaptive Server 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.

Support for OData

Adaptive Server version 15.7 SP100 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 Adaptive Server. 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.

See also

- *OData Server Sample Files* on page 94

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.

See also

- *Security Considerations for OData Server* on page 88

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.

See also

- *Configuring OData Server* on page 89
- *Starting and Stopping OData Server* on page 95

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-filename***	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.

Option	Description
ShutdownListenerPort = *port_number*	Specifies the port number on which the embedded server listens for shutdown requests. The default setting is 2449.
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.
ConnectionPoolMaximum = *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.
DbConnectionString = *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 = ***connec-tion-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 Adaptive Server 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.

See also

- *Configuring OData Server* on page 89
- *Starting and Stopping OData Server* on page 95

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

Adaptive Server 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 Adaptive Server 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 Adaptive Server 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 Adaptive Server installer creates all the folders and copies the files necessary to run OData. After you successfully install Adaptive Server, 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.

Command	Description
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 ShutdownListener-Port 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.

See also

- *Configuring OData Server* on page 89

MIT Kerberos and NTLM Security Services

MIT Kerberos and NT Lan Manager security services are supported on Windows 64-bit.

Changes to the `libtcl64.cfg`, `libtcl.cfg`, and `sql.ini` files are required to use these security services.

Using NT Lan Manager Security Services on Windows 64-bit

NT Lan Manager security services are supported on Windows 64-bit.

In order to use the NTLM Security Services on Windows 64bit, changes are required in the `libtcl64.cfg`, `libtcl.cfg`, and `sql.ini` files.

1. Update the `[SECURITY]` section of the `%SYBASE%\%SYBASE_OCS%\ini\libtcl64.cfg` file.
   a) Add the following to the `[SECURITY]` section:
   ```
   NTLM=LIBSYBSMSSP64
   ```
   This change allows 64-bit ASE and 64-bit Open Client applications to use the security driver library `libsybsmssp64.dll` at runtime. The `libsybsmssp64.dll` library is located in `%SYBASE%\%SYBASE_OCS%\dll`, along with other Open Client dynamic link libraries.
2. Update the `[SECURITY]` section in the `%SYBASE%\%SYBASE_OCS%\ini\libtcl.cfg` file.
   a) Add the following to the `[SECURITY]` section:
   ```
   NTLM=LIBSYBSMSSP
   ```
   The `libtcl.cfg` is used by the 32bit `isql` utility and 32-bit OpenClient applications.
3. Choose one of the following methods to specify the OID value for NTLM.
   - Update the `[SECMECH]` section of the `%SYBASE%\ini\sql.ini` file. Add the following to the `sql.ini` file:
     ```
     [ASENAME]
     master=TCP,<host>,<port>
     query=TCP,<host>,<port>
     secmech=1.3.6.1.4.1.897.4.6.3
     ```
   - Use the `dsedit` utility to add the 'Server Security' attribute value of `'1.3.6.1.4.1.897.4.6.3'` to your server.

   Note: This OID value derived from the `%SYBASE%\ini\objectid.dat` file which should not be modified

.

4. Make sure Adaptive Server is configured for security services. For example, to enable services with LAN Manager, execute:

   ```
   sp_configure "use security services", 1
   ```

   For more information, see *Using Security Services with NT LAN Manager* in the *Configuration Guide for Windows NT*.
5. Make sure you have a login on the Adaptive Server that corresponds with your Windows login.
6. Connect to the Adaptive Server without a user name and password. For example:
   - `isql -V -SASENAME`
   - `isql64 -V -SASENAME`

Using MIT Kerberos Security Services on Windows 64-bit

MIT Kerberos security services are supported on Windows 64-bit.

In order to use the MIT Kerberos Security services on Windows 64bit, changes are required in `libtcl64.cfg`, `libtcl.cfg`, and `sql.ini` files.

1. Update the `[SECURITY]` section of the `%SYBASE%\%SYBASE_OCS%\ini\libtcl64.cfg` file.

   a) Add the following to the `[SECURITY]` section:

   ```
   csfkrb5=LIBSYBSKRB64 secbase=@REALM
   libgss=MIT_KRB_64_INSTALL_DIR \bin\gssapi64.dll
   ```

   Note: For the above example:

   REALM should be replaced with the Kerberos realm name.

   MIT_KRB_64_INSTALL_DIR should be replaced with the directory where MIT Kerberos version 4.0.1 for Windows 64-bit is installed.

   The path to the `gssapi` library, used in the `libtcl64.cfg` file, cannot contain whitespaces.

   This change allows 64-bit ASE and 64-bit Open Client applications to use the security driver library `libsybsmssp64.dll` at runtime. The `libsybsmssp64.dll` library is located in `%SYBASE%\%SYBASE_OCS%\dll`, along with other Open Client dynamic link libraries.
2. Update the `[SECURITY]` section in the `%SYBASE%\%SYBASE_OCS%\ini\libtcl.cfg` file.

   a) Add the following to the `[SECURITY]` section:

```
 csfkrb5=LIBSYBSKRB secbase=@REALM
libgss=MIT_KRB_32_INSTALL_DIR   \bin\gssapi32.dll
```

Note: For the above example:

REALM should be replaced with the Kerberos realm name.

MIT_KRB_32_INSTALL_DIR should be replaced with the directory where MIT Kerberos version 4.0.1 for Windows 32-bit is installed.

The path to the `gssapi` library, used in the `libtcl64.cfg` file, cannot contain whitespaces.

The `libtcl.cfg` is used by the 32bit `isql` utility and 32-bit OpenClient applications.

3. Choose one of the following methods to specify the OID value for MIT Kerberos.

- Update the `[SECMECH]` section of the `%SYBASE%\ini\sql.ini` file. Add the following to the `sql.ini` file:

```
[ASENAME]
master=TCP,<host>,<port>
query=TCP,<host>,<port>
secmech=1.3.6.1.4.1.897.4.6.6
```

- Use the `dsedit` utility to add the 'Server Security' attribute value of `'1.3.6.1.4.1.897.4.6.6'` to your server.

Note: This OID value derived from the `%SYBASE%\ini\objectid.dat` file which should not be modified

.

4. Make sure Adaptive Server is configured for security services. For example, to enable services with LAN Manager, execute:

```
sp_configure "use security services", 1
```

For more information, see *Using Security Services with NT LAN Manager* in the *Configuration Guide for Windows NT*.

5. Connect to the Adaptive Server without a user name and password. For example:

- `isql -V -SASENAME`
- `isql64 -V -SASENAME`

Solaris Asynchronous I/O Performance

A new configuration parameter has been added to address performance issues for applications running in threaded kernel mode against Adaptive Server on the Solaris platform.

The **solaris async i/o mode**, along with the Solaris patch containing the fix for Oracle BugID 16054425, can improve performance for applications running against Adaptive Server on the Solaris platform in threaded mode.

See also

- *New Configuration Parameters* on page 145

Properties Added to Responses File

New properties have been added to the Adaptive Server responses file for all platforms.

Property	Value	Description
DO_UPDATE_ASE_SERVER	true or false	If true, updates the existing Adaptive Server instance. This property is valid only if DO_UPDATE_INSTALL is true.
UPDATE_ASE_SERVER_NAME_[n]	Server name	Adaptive Server name to update
UPDATE_ASE_PASSWORD_[n]	Adaptive Server security administer password	Adaptive Server sa password

For more information, see the *Adaptive Server Upgrades* in the installation guide for your platform.

System Changes

Adaptive Server 15.7 SP100 adds changes to commands, system procedures, functions, monitoring tables, and utilities.

New Functions

New Adaptive Server 15.7 SP100 functions.

defrag_status

Returns metrics of any defragmentation operation that is started or ongoing on the named object or partition.

Syntax

```
defrag_status( dbid, objjid [ , ptnid | -1 [, "tag" ] ]
```

Parameters

- ***dbid* –** is the ID of the target database.
- ***objid* –** is the ID of the target object.
- ***ptnid* –** is the ID of the partition or enter `-1`.

  `-1` refers to all the partitions in the table. If *ptnid* is unspecified, `-1` is the default value.

  In case of invoking the built-in with four parameters, the third parameter

  'ptnid' cannot be skipped. So, it has to be specified accordingly.

- ***tag* –** is one of:
  - **frag index** or **fragmentation index**– the fragmentation index is the number of times the size of the object is larger compared to the size of the same if it was completely defragmented.

    This index can be any number greater than or equal to zero. The lower the index, the less fragmented the table or partition is. The higher the index, the more fragmented the object is and is more likely to free up space with defragmentation.

    For example, a value of 0.2 , means the table occupies 20% more space than what it would be if the data were fully defragmented. This index can be any number > 0. For example, 1 means the table is occupying 100% more space than what a fully defragmented version of the data would occupy.

- **pct defrag** or **pct defragmented** – is the percentage of pages defragmented.
- **pages defrag** or **pages defragmented** – the number of pages defragmented.
- **pages gen** or **pages generated**– the number of new pages generated.
- **pages tbd** or **pages to be defragmented**– the number of pages still left to be processed and defragmented.
- **last run** – the start time of the most recent invocation of this command.
- **executing** – boolean, whether the command is executing currently.
- **elapsed mins** – the number of minutes elapsed since the start of the most recent invocation of this command. This value is non-zero when **executing** is 1, and is zero otherwise.

Examples

- **Example 1 –** executes **defrag_status** on the table mymsgs:

```
select defrag_status(db_id(), object_id('mymsgs'))
```

If defragmentation has not yet been performed, the output is:

```
------------------------------------------------------------------
frag index=0.20, pct defrag=0, pages defrag=0, pages gen=0,
pages tbd=1174, last run=, executing=0, elapsed mins=0
```

If defragmentation has been performed, the output is:

```
-----------------------------------------------------------------------
frag index=0.07, pct defrag=100, pages defrag=1167, pages gen=1072,
pages tbd=0, last run=Oct  9 2012  2:27:11:446PM, executing=0,
elapsed mins=0
```

- **Example 2 –** executes **defrag_status** on the data partition p1:

```
select defrag_status(db_id(), object_id('t1'), partition_id('t1', 'p1'))
```

If defragmentation has not yet been performed, the output is:

```
-----------------------------------------------------------------------
frag index=0.75, pct defrag=0, pages defrag=0, pages gen=0, pages tbd=67,
last run=, executing=0, elapsed mins=0
```

If defragmentation is executed, the output is:

```
------------------------------------------------------------------------
frag index=0.00, pct defrag=100, pages defrag=61, pages gen=32,
pages tbd=0, last run=Oct  9 2012  2:44:53:830PM, executing=0,
elapsed mins=0
```

If partial defragmentation is executed, the output is:

```
-----------------------------------------------------------------------
frag index=0.02, pct defrag=41, pages defrag=135, pages gen=144,
pages tbd=190, last run=Oct 9 2012  3:17:56:070PM, executing=0,
elapsed mins=0
```

While defragmentation is in progress, the output is:

```
---------------------------------------------------------------------
frag index=0.90, pct defrag=10, pages defrag=40, pages gen=24,
pages tbd=360, last run=Oct 9 2012  3:01:01:233PM, executing=1,
elapsed mins=1
```

- **Example 3 –** executes the **pct defrag** parameter:

  ```
  select defrag_status(db_id(), object_id('t1'), -1, 'pct defrag')
  ```

  The output displays the percentage of the pages that have been defragmented.

```
-----------------------------------------------------------------
8
```

When 1 row is affected:

```
select defrag_status(db_id(), object_id('t1'), partition_id('t1', 'p1'),
   'pct defrag')
```

The output is:

```
------------------------------------------------------------------
41
```

See also

- *Checking the Reorganization Status* on page 41

loginfo

Returns information about a transaction log

Syntax

```
loginfo (dbid | dbname, option]
```

Parameters

- ***dbid* –** is the database ID.
- ***dbname* –** is the database name.
- ***option* –** is the specific information about the log you need. Options are:
  - **help** – shows a message with the different options.
  - **first_page** – returns the page number of the first log page.
  - **root_page** – returns the page number of the last log page.
  - **stp_page** – returns the page number of the secondary truncation point (STP), if it exists. The secondary truncation point (or STP) is the point in the log of the oldest transaction yet to be processed for replication. The transaction may or may not be active. In cases where the transaction is no longer active, the STP by definition precedes the oldest active transaction.
  - **checkpoint_page** – returns the page number in the log that contains the most recent checkpoint log record.

- **checkpoint_marker** – returns the record ID (RID) in the log that contains the most recent checkpoint log record.
- **checkpoint_date** – returns the date of the most recent checkpoint log record.
- **oldest_transaction_page** – returns the page number in the log on which the oldest active transaction at the time of the most recent checkpoint, started. If there was no active transaction at the time of the most recent checkpoint, **oldest_transaction_page** returns the same value as **checkpoint_page**.
- **oldest_transaction_marker** – returns the RID (page number and row ID) in the log on which the oldest active transaction at the time of the most recent checkpoint, started. If there was no active transaction at the time of the most recent checkpoint, **oldest_transaction_marker** returns the same value as **checkpoint_marker**.
- **oldest_transaction_date** – is the at which the oldest active transaction started.
- **until_time_date** – is the latest time that could be encapsulated in the dump that is usable by the **until_time** clause of **load transaction**.
- **until_time_page** – is the log page on which the log record associated with **until_time_date** resides.
- **until_instant_marker** – is the RID (page number and row ID) of the log record associated with **until_time_date**.
- **total_pages** – is the total number of log pages in the log chain, from **first_page** to **root_page**.
- **stp_pages** – the total number of log pages between the STP and the oldest active transaction.
- **active_pages** – the total number of pages between the oldest transaction at the time of the most recent checkpoint, and the end of the log.
- **inactive_pages** – the total number of log pages between **first_page** and either **stp_page** or **oldest_transaction**, whichever comes first. This is the number of log pages that will be truncated by the **dump transaction** command.

Examples

- The result type of **loginfo** is `bigint`, which means you must convert the output when your query results in a RID or a date.

Permissions

The user must have sa_role to execute **loginfo**.

show_condensed_text

Returns the unified SQL text for cached statements.

Syntax

```
show_condensed_text(statement_id, option)
```

Parameters

- ***statement_id* –** is ID of the statement. Derived from the SSQLID column of `monCachedStatement`.
- ***option* –** is a string constant, enclosed in quotes. One of:
  - text – returns the condensed text
  - hash – return the hash value for the condensed text

Examples

- **Example 1 –** displays condensed text for cached SQL text:

  ```
  select show_condensed_text(SSQLID, 'text') from
  monCachedStatement

   -------------------------------------------------------------
   SELECT SHOW_CONDENSED_TEXT(SSQLID,$) FROM monCachedStatement
  ```
- **Example 2 –** displays the hash value of the condensed text for cached SQL text: 1:

  ```
   select show_condensed_text(SSQLID, 'hash') from
  monCachedStatement

   -------------------------------------------------------------
  1331016445
  ```

Usage

show_condensed_text:

- Returns a `text` datatype
- Supports long SQL text (greater than 16KB)
- Returns NULL for invalid *option* values

Permissions

The permission checks for **show_condensed_text** depend on your granular permissions settings:

- Granular permissions enabled – you must have the `mon_role`, or have `monitor qp performance` permission to execute **show_condensed_text**.
- Granular permissions disabled – you must have the `mon_role` or `sa_role` to execute **show_condensed_text**.

shrinkdb_status

Determines the status of a shrink operation.

Syntax

```
 shrinkdb_status(database_name, query)
```

Parameters

- **database_name –** is the name of the database you are checking.
- **query –** is one of:
  - **in_progress** – determines if a shrink database is in progress on this database. Returns a value of 0 for no, a value of 1 for yes.
  - **owner_instance** – determines which instance in a cluster is running a shrink operation. Returns:
    - 0 – if no shrink is in progress.
    - The owning instance ID – if an instance has a shrink operation running. For a nonclustered server, the "owning instance” is always 1.
  - **au_total** – returns the total number of allocation units (that is, groups of 256 pages) the shrink operation affects.
  - **au_current** – returns the total number of allocation units processed by the shrink operation.
  - **pages_moved** – returns the number of index or data pages moved during the current shrink operation. **pages_moved** does not include empty pages that were released during the shrink operation.
  - **begin_date** – the date and time the current shrink operation began, returned as an `unsigned bigint`.
  - **end_date** – returns the date and time the shrink operation ended. Returns 0 when the shrink operation is ongoing or completed but not waiting for a restart.
  - **requested_end_date** – returns the date and time the active shrink operation is requested to end.
  - **time_move** – returns the amount of time, in microseconds, spent moving pages. **time_move** includes the time spent updating page references to the moved pages, but does not include the time spent performing administrative tasks that happen at the end of individual move blocks.
  - **time_repair** – returns the amount of time, in microseconds, spent on administrative tasks for moving blocks. **time_repair** plus the value for **time_move** indicates the approximate amount of time Adaptive Server spent working on the current shrink operation.
  - **last_error** – returns the error the shrink operation encountered when it came to abnormal stop.

Examples

- **Example 1 –** checks the progress of the `pubs2` database shrink operation:

```
shrinkdb_status("pubs2", "in_progress")
```

- **Example 2 –** returns the amount of time Adaptive Server spent moving the pages of the `pubs2` database:

```
shrinkdb_status("pubs2", "time_move")
```

- **Example 3 –** returns the amount of time Adaptive Server spent shinking the `pubs2` database:

  ```
  shrinkdb_status("pubs2", "time_move")
  ```

Usage

shrinkdb_status returns 0 if no shrink operations are currently running on the database.

Changed Commands

Adaptive Server 15.7 SP100 includes changes to commands.

alter database

Adaptive Server 15.7 SP100 includes changes to the **alter database** command.

off

alter database allows you to shrink databases, and adds the **off** parameter, which specifies the device names from which you are releasing space. The syntax is:

```
alter database database_name
. . .
     off database_device {=size  | [from page_number] [to
page_number]}
     [, database_device…]
     [with timeout='time']
```

create index

Adaptive Server 15.7 SP100 includes changes to the **create index** command.

online

create index adds the online parameter, which lets you create indexes without blocking access to the data you are indexing. The syntax is:

```
create [unique] [clustered | nonclustered] index index_name
    on database.]owner.]table_name
    [with {...
            online,
    ...}
```

sorted_data

You may execute parallel **create index**:

- For a clustered index on a data-only-locked table (also called a placement index)
- With the **sorted_data** clause

create index with sorted_data selects a parallel query plan only if you include an explicit **consumers =** clause. For example, Adaptive Server uses a parallel query plan for the first query, but uses a serial query plan for the second:

```
create index i1 on t1(c1) with sorted_data, consumers = N
create index i1 on t1(c1) with sorted_data
```

Note: Running query plans in serial allows Adaptive Server to dump the transaction log, maintaining recoverability. Running **create index** in parallel may cause the log records to become out of order, and you may not be able to dump the transaction log. Instead you must dump the database to maintain recoverability.

with statistics {no_hashing | *new_hashing* | *hashing}]*

Use the hash-based option and additional hash option clauses to gather index statistics on tables.

The **with statistics hashing** clauses are:

- **max_resource_granularity**
- **histogram_tuning_factor**
- **print_progress**

The syntax is:

```
create [unique] [clustered | nonclustered] index index_name
    on database.]owner.]table_name
    [with {...
            [, statistics { no_hashing | new_hashing |
hashing }]
            [, statistics max_resource_granularity = int]
           [, statistics histogram_tuning_factor = int]
            [, statistics print_progress = int] } ]
    ...}
```

See also

- *Gathering Hash-Based Statistics with create index* on page 63
- *sp_configure* on page 123

create materialized view

Adaptive Server 15.7 SP100 changes the **create materialized view** command.

You cannot include a **like** clause with **create precomputed result set** or **create materialized view** commands that include an **immediate refresh** parameter.

create precomputed result set

Adaptive Server 15.7 SP100 changes the **create precomputed result set** commands.

You cannot include a **like** clause with **create precomputed result set** or **create materialized view** commands that include an **immediate refresh** parameter.

dbcc

Adaptive Server 15.7 SP100 includes changes to the **dbcc** command.

zapdefraginfo

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.

The syntax is:

```
dbcc zapdefraginfo( dbid | dbname, objid | objname
[, indexid | indexname
[,  ptnid | ptnname | 0,
['print' | 'zap' | 'zap_no_log' ]]])
```

where:

- **zapdefraginfo** – deletes incremental reorganization information from `sysattributes` and the corresponding meta-information on allocation pages.
- *dbid* **|** *dbname* – specifies the target database ID or name and deletes the `sysattributes` rows pertaining to incremental reorganization under the class "misc table info."
- database_name – is the name of the database to check. If no database name is given, dbcc uses the current database.
- *indexid* **|** *indexname* – specifies the ID or name of the index of the target partition. Currently the value '0' for `indexid` or the name of the table for `indexname` are supported in this parameter.
- *objid* **|** *objname* – specifies the ID or name of the accessed object.
- *ptnid* **|** *ptnname* – specifies the ID or name of the target partition and deletes only the extents that correspond to that partition. The value of 0 for this parameter specifies that all the partitions of the object must be considered.
- **print** – prints each extent and the extent version of the specified object or partition.
- **zap** – deletes the extent version information for the specified object or partition.
- **zap_no_log** – deletes the extent version information for the specified object without logging the changes involved.

In cases where the `sysattributes` incremental reorganization rows are accidentally deleted, use **dbcc zapdefraginfo** to reset the extent version information of the object so that **reorg defrag** does not mistakenly skip the supposedly fragmented extents while reorganizing.

shrinkdb_setup

Checks whether backlink pointers exist for each partition of a table in the database. If the backlink pointers do not exist, **shrinkdb_setup** creates them. The syntax is:

```
dbcc shrinkdb_setup(dbname[,object_name[, force [, verbose]]]
```

where:

- *dbname* – is the name of the database you are checking.
- *object_name* – is the name of the object you are checking.
- **force** – is:
  - **true** – check tables and partitions whether or not they are already marked as being maintained.
  - **false** – do not check the tables or partitions.
- **verbose** – print messages at the beginning of each check performed (on by default).

Usage:

- After you run **dbcc shinkdb_setup**
  - Mark any large object columns that include replication indexes as "suspect" to ensure that Replication Server uses the backlink pointers on these LOB columns.
  - Databases are marked to indicate that **dbcc shinkdb_setup** has configured backlink pointers for text and image columns. You cannot run a **use index** command to create replication indexes.

dbrepair

Repairs corruption caused by a shrink database operation that stopped due to an uncontrolled situation.

The syntax is:

```
dbcc dbrepair(dbname, redo_shrink)
```

where:

- *dbname* – is the name of the database you are checking.
- **redo_shrink** – indicates that you are repairing a database that suffered an uncontrolled halt.

dump database

Adaptive Server 15.7 SP100 includes changes to the **dump database** command to support the cumulative dump feature.

The **dump database** command includes a new **cumulative** type, which allows you to specify that the backup you create is a cumulative incremental dump. The new command syntax is:

```
 dump database database_name cumulative
   using config[uration] = config_name
   to [compress::[compression_level::]]stripe_device
    [at backup_server_name]
    [density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    dumpvolume = volume_name,
    file = file_name]
    [[stripe on [compress::[compression_level::]]stripe_device
    [at backup_server_name]
    ...]...]
    [with { density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    compression = compress_level,
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    retaindays = number_days,
    [noinit | init],
    notify = {client | operator_console} }]
```

The **dump database cumulative** command supports the same options as **dump database**.

To perform a full backup, you may specify **dump database** *database_name* **full**, however, **full** is optional. If you do not specify dump type, **dump database** assumes you are performing a full backup.

See also

- *Cumulative Dump* on page 13
- *load database* on page 115

load database

Adaptive Server 15.7 SP100 includes changes to the **load database** command.

The **load database** command includes a new **cumulative** type, which allows you to load a backup created with the **dump database cumulative** keyword. The new command syntax is:

```
 load database database_name cumulative
   using config[uration] = config_name
   to [compress::[compression_level::]]stripe_device
    [at backup_server_name]
    [density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    dumpvolume = volume_name,
    file = file_name]
    [[stripe on [compress::[compression_level::]]stripe_device
    [at backup_server_name]
    ...]...]
    [with { density = density_value,
    blocksize = number_bytes,
    capacity = number_kilobytes,
    compression = compress_level,
    dumpvolume = volume_name,
    file = file_name,
    [dismount | nodismount],
    [nounload | unload],
    retaindays = number_days,
    [noinit | init],
    notify = {client | operator_console} }]
```

The **load database cumulative** command supports the same options as **load database**.

To perform a full restoration, you may specify **load database** *database_name* **full**, however, **full** is optional. If you do not specify dump type, **load database** assumes you are performing a full backup.

See also

- *Cumulative Dump* on page 13
- *dump database* on page 114

reorg

The **defrag** parameter of the **reorg** command lets you schedule and resume reorganization, while also allowing concurrent reads or writes on the data being reorganized.

The syntax is:

```
reorg defrag table_name [partition {partition_list}]
   [with {time = hh:mm | resume | skip_compact_extents
   [= pct_value]}]
```

where:

- **defrag** – reorganizes each partition list or partition in the table while allowing concurrent reads or writes on the data being reorganized.
- **partition** – is the subset of the table that shares the same column definitions, constraints, and defaults as the table.
- *partition_list* – is the list of partition names.
- **time** – reorganizes the table or the list of partitions for the specified interval of time. *hh* is the number of hours and has no limit, and *mm* is the number of minutes 0–59.
- **resume** – resumes table reorganization from the end of the last reorganized data page invoked from the previous **reorg defrag**. **resume** continues until the entire table or list of partitions is reorganized. Running **time = *hh:mm*** with **resume** indicates that you are running reorganization from the previous position of reorganization and running it only for the specified interval of time.
- **with skip_compact_extents** – skips compact extents. The compactness of an extent is measured as the percentage occupancy in that extent.
- *pct_value* – is the compactness of an extent measured as the percentage occupancy in that extent with a value of 1–100.

  Compactness = (Total space occupied in an extent / Total space in an extent) x 100.

  If **with skip_compact_extents** is used, all the extents with compactness greater than or equal to the threshold occupancy percent value specified would be skipped for reorganization. If no threshold percent value is specified, the default percent value is 80%.

See also

- *Running reorg defrag* on page 39

select

Adaptive Server 15.7 SP100 includes changes to the **select** command.

ins_by_bulk

The **ins_by_bulk** parameter improves performance by directly inserting data rows into newly allocated data pages by bulk for tables. You can set **ins_by_bulk** at the query level using the abstract plan for a specific insert statement.

The syntax is:

```
insert into. . .
select . . .
plan "(use ins_by_bulk on)"
```

For example:

```
insert into my_salesdetail (stor_id, ord_num, title_id, qty,
discount)
select stor_id, ord_num, title_id, qty, discount from salesdetail
where qty > 100
plan '(use ins_by_bulk on)'
```

See also

- *set* on page 117
- *Enable Data Load Optimization* on page 35

set

Adaptive Server 15.7 SP100 includes changes to the **set** command.

set Option	Description
ins_by_bulk	The **set** command enables the **ins_by_bulk** parameter for the session, which improves performance by directly inserting data rows into newly allocated data pages for tables. The syntax is: `set ins_by_bulk {on | off}`
join_bloom_filter	The **join_bloom_filter** parameter enables or disables the use of bloom filters for query plan optimization. The syntax is: `set join_bloom_filter {on | off}`
statistics parallel_plan_detail_html	The **parallel_plan_detail_html** parameter generates a graphical query plan in HTML format containing information about details per thread and plan fragments for query plans that are executed in parallel using several worked threads. Use this option to diagnose the behavior of specific threads and plan fragments in comparison with the global execution of the query. The syntax is: `set statistics parallel_ plan_detail_html {on | off}`

set Option	Description
statistics plan_detail_html	The **plan_detail_html** parameter generates a graphical query plan in HTML format containing information details of plan operators, such as the name, different timestamps captured during the execution, number of rows affected, number of estimated rows, elapsed time, and so on. The syntax is: `set statistics plan_detail_html {on | off}`
statistics plan_directory_html	The **plan_directory_html** parameter specifies the directory path name into which to write the HTML query plans. The file name is identified by a combination of user name, spid, and timestamp. The syntax is: `set statistics plan_directory_html {dirName | on | off}` When set to **off**, the dumping of the HTML data to an external file is stopped. When set to **on**, the dumping of HTML data to an external file in a directory previously indicated is resumed. No output is generated if a directory name was not previously provided.
statistics plan_html	The **plan_html** parameter generates a graphical query plan in HTML format containing information about the number of rows and number of threads per operator. The syntax is: `set statistics plan_html {on | off}`
statistics timing_html	The **timing_html** parameter generates a graphical query plan in HTML format containing execution statistics related to the timing spent in each operator per execution phase. CPU usage and Wait distribution is generated for queries executed in parallel. The syntax is: `set statistics timing_html {on | off}`

See also

- *Query Detail* on page 56
- *Query Plan Tree Structure* on page 48
- *set statistics Command Options* on page 48
- *Parallel Query Plan Fragments* on page 58
- *Writing HTML to an External File* on page 61
- *select* on page 117
- *Enable Data Load Optimization* on page 35
- *Batch Insert Optimization* on page 37
- *Improved Data Load Performance* on page 35
- *Query Plan Optimization with Bloom Filters* on page 67

update statistics

Adaptive Server 15.7 SP100 changes the **update statistics** command.

update statistics acquires memory from the default data cache instead of `tempdb` buffer cache if the session is using a `tempdb` bound to an inmemory device.

Procedures

Adaptive Server 15.7 SP100 contains new and changed system procedures.

New System Procedures

New Adaptive Server 15.7 SP100 system procedures.

sp_helpdefrag

sp_helpdefrag reports defragmentation information for either all eligible objects for **reorg defrag** in the database whose context it is invoked from or for the given object if it is eligible for **reorg defrag**.

sp_helpdefrag uses the built-in **defrag_status()** on each of the required tables or on each of the required data partitions to get the information about defragmentation.

- If *table_name* is not specified, defragmentation information for all eligible tables for *reorg defrag* (that is, user tables with datarows or datapages locking scheme) is reported. Rows for tables on which *reorg defrag* is currently executing precede those for tables where *reorg defrag* is not currently executing. Among these two sets, rows are in ascending order of the **pct_defrag**.
- If *table_name* is specified, and if the table is eligible for *reorg defrag*, defragmentation information of the table as well as that of each data partition is reported. Rows are in the ascending order of percentage defragmented portion. Row for the table comes first and has `NULL` in partition column.
- If *partition_name* is specified, only that particular data partition's information is reported.

Syntax

The syntax is:

```
sp_helpdefrag [table_name][,partition_name]
```

Parameters

- ***table_name*** **–** is the name of the table.
- ***partition_name*** **–** is the name of the partition.

Examples

- **No parameters and before defragmentation –** If **sp_helpdefrag** is executed without parameters on database testdb with user data-only locking tables before defragmentation:

  ```
  sp_helpdefrag
  ```

  The output is:

```
table             frag_index     pct_defrag    executing      last_run
---------------- ------------- ------------- -------------- ----------
t1_forw                 0.01           0                0          NULL
mymsgs                  0.39           0                0          NULL
mymsgs_clone            0.57           0                0          NULL
t1                      0.66           0                0          NULL
myprocs                 0.86           0                0          NULL
mymsgs_ptnd             1.07           0                0          NULL
t1_clone                1.98           0                0          NULL
myprocs_clone           2.16           0                0          NULL
t1_ptnd                 2.99           0                0          NULL
myprocs_ptnd            3.03           0                0          NULL

(1 row affected)
(return status = 0)
```

  If you execute **sp_helpdefrag** after defragmentation, the output is:

```
table           frag_index    pct_defrag    executing         last_run
------------- ------------- ------------- ------------- --------------
t1_forw          0.01         100           0         Oct 10 2012  4:15PM
mymsgs           0.05         100           0         Oct 10 2012  4:15PM
mymsgs_clone    0.06         100            0         Oct 10 2012  4:15PM
t1              0.08         100           0          Oct 10 2012  4:15PM
myprocs          0.09         100           0         Oct 10 2012  4:15PM
mymsgs_ptnd     0.09         100            0         Oct 10 2012  4:15PM
t1_clone         0.10         100          0          Oct 10 2012  4:15PM
myprocs_clone   0.11         100            0         Oct 10 2012  4:15PM
t1_ptnd          0.12         100           0         Oct 10 2012  4:15PM
myprocs_ptnd      0.14         100          0         Oct 10 2012  4:15PM

(1 row affected)
(return status = 0)
```

- **On a specified table –** If **sp_helpdefrag** is executed on table t1 in database testdb:

  ```
  sp_helpdefrag t1
  ```

  The output is:

```
table   partition  frag_index   pct_defrag  executing         last_run
------- ---------- ----------- ------------- ---------- ------------------
t1       NULL       0.35          35          0      Oct 10 2012  4:33PM
t1       p2          0.50          0           0                     NULL
t1       p1         0.42          20          0      Oct 10 2012  4:33PM
t1       p3         0.42          20          0      Oct 10 2012  4:33PM
t1       p4         0.05         100          0      Oct 10 2012  4:33PM

(1 row affected)
(return status = 0)
```

If **reorg defrag** is currently processing, the output is:

```
table   partition  frag_index   pct_defrag  executing         last_run
------- ---------- ----------- ------------- ---------- -----------------
t1       NULL        0.48          13          1      Oct 10 2012  4:33PM
t1        p2        0.50            0         1                      NULL
t1       p4         0.60            0          1      Oct 10 2012  4:33PM
t1       p1         0.42           20          1      Oct 10 2012  4:33PM
t1       p3         0.42           20          1      Oct 10 2012  4:33PM

(1 row affected)
(return status = 0)
```

- **On a specified partition –** If **sp_helpdefrag** is executed on partition p1 in table t1:

  ```
  sp_helpdefrag t1, p1
  ```

  The output is:

```
table   partition  frag_index   pct_defrag  executing       last_run
------- ---------- ----------- ------------- ---------- -----------------
t1        p1        0.42           20          0      Oct 10 2012  4:33PM

(1 row affected)
(return status = 0)
```

See also

- *Checking the Reorganization Status* on page 41

sp_dump_info

The **sp_dump_info** system procedure displays the size of data and log that an uncompressed cumulative dump would contain at a specific point in time.

The size is reported in units of KB, MB, or GB, as appropriate. The size reported may be slightly smaller than the actual size of the archive file (or files, if using multiple stripes), because the archive contains some additional information by way of the labels, header, trailer and runlist pages. **sp_dump_info** can also only assume that an uncompressed dump is done; if a compressed dump is done, the size of the archive will clearly be smaller than that reported by **sp_dump_info**.

You cannot use **sp_dump_info**:

- Unless you allow incremental dumps of your database by using the **allow incremental dumps** parameter of **sp_dboption**.
- If the database has not yet been fully dumped since you enabled incremental dumps for your database.

Syntax

```
sp_dump_info database_name
```

Parameters

- **database_name –** is the name of the database.

Examples

- **Data and log size –** Displays the size of data and log that the cumulative dump of the `test` database contains

```
sp_dump_info test
go
 Data     Log    Database percentage  Allocation threshold
 -------  -----  -------------------  --------------------
 4368 KB  2 KB                     2                    40
(return status = 0) (return status = 0)
```

  The output indicates that if a cumulative dump were taken at this point in time, it would contain approximately 4,368KB of data and a single log page, which represents 2 percent of the total database size.

  Compare this with the size if you performed a cumulative dump at this time:

```
dump database test cumulative to "c:/tmp/test.dmp"
go
Backup Server: 4.171.1.1: The current value of 'reserved pages
threshold' is 85%.
Backup Server: 4.171.1.2: The current value of 'allocated pages
threshold' is 40%.
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 'test122480F0EF ' section
number 1 mounted on disk file 'c:/tmp/test.dmp'
Backup Server: 4.188.1.1: Database test: 4328 kilobytes (3%)
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 test: 4370 kilobytes (3%)
DUMPED.
Backup Server: 3.42.1.1: DUMP is complete (database test).
```

  The corresponding size of the archive is 4,487,168 bytes, or 2191 pages. This differs from the estimate given by **sp_dump_info** by 29 pages (14 KB), which is the result of 8 pages for the dump labels, 1 page for the dump header, 1 page for the dump trailer and 19 pages containing run lists. The size of the dump labels, header and trailer are independent of the numbers of pages dumped, while the number of pages used by run lists is dependent on the numbers of pages dumped.

- **Error message –** Displays an error message when incremental dumps are not enabled on `master`

```
 sp_dump_info mydb
go
Msg 17154, Level 16, State 1:
```

```
Procedure 'sp_dump_info', Line 32:
Incremental dumps are not enabled in database mydb.
(return status = 1)
```

Usage

sp_dump_info fails if you do not allow incremental dumps, or you have not enabled incremental dumps for your database.

Permissions

Any user can execute **sp_dump_info**.

See also

- *Cumulative Dump* on page 13

Changed System Procedures

Changed Adaptive Server 15.7 SP100 system procedures.

sp_configure

The **sp_configure** system procedure supports features introduced in Adaptive Server 15.7 SP100.

optimize temp table resolution

Summary information	
Default value	0
Range of values	0, 1
Status	Dynamic
Display level	Intermediate
Required role	System administrator
Configuration groups	Query Tuning

When enabled, allows Adaptive Server to reuse the cached statement plan for multiple sessions, even if the statement involves temporary tables. Adaptive Server reuses the cache statement plan if the schema for the recreated temporary tables is the same as the previous temporary tables.

Cumulative Dumps

With the introduction of cumulative dumps, you can now configure these settings using **sp_configure**:

- You can now use the **sp_configure** system procedure to configure Adaptive Server to enable concurrent transaction dumps and cumulative database dumps by setting the **enable concurrent dump tran** parameter 1:

  ```
  sp_configure 'enable concurrent dump tran', 1
  ```

  You cannot, however, have concurrent **dump cumulative** and **dump database**, since **dump cumulative** needs **dump database** to finish, because **dump cumulative** copies all those pages that changed once **dump database** is finished.
- Record cumulative dump commands in the dump history file by enabling the **enable dump history** configuration parameter:

  ```
  sp_configure 'enable dump history', 1
  ```

Gathering Index Attributes Using Hash-Based Statistics

The **utility statistics hashing** option for **sp_configure** is available in Adaptive Server 15.7 SP100.

utility statistics hashing enables the gathering of index attributes using hash-based statistics when creating an index. The options are:

- **on** – index attributes are gathered using hash-based statistics.
- **off** – the sort-based algorithm from versions of Adaptive Server earlier than 15.7 ESD #2 is used.
- **new** – hashing is gathered for minor attributed columns that have not had statistics previously gathered.
- **default** – same as **off**.

The values of **create index** hashing option take precedence over **utility statistics hashing**. If the clauses **hashing**, **new hashing**, or **no_hashing** have not been specified in the **create index** command, the value of **utility statistics hashing** is used.

See also

- *Gathering Hash-Based Statistics with create index* on page 63
- *create index* on page 111

sp_config_rep_agent

The **sp_config_rep_agent** system procedure has been updated to support the additional distribution model **filter** for the **multipath distribution model** property.

Select the **filter** distribution model to distribute data from the primary database through the available replication paths to achieve parallel replication and improved replication performance. Enable multiple scanners if you intend to set the distribution model to **filter**.

To set the distribution model:

```
sp_config_rep_agent database, ‘multipath distribution model’,
{‘connection' | 'object' | 'filter'}
```

- **multipath distribution model** – is the distribution model parameter for **sp_config_rep_agent**
- **connection** – sets the model to distribution by connection
- **filter** – sets the model to distribution by column filter

For information about setting the distribution model and configuring the distribution by column filter, see *Additional Distribution Mode for Multi-Path Replication* in the *Replication Server New Features Guide*.

See also

- *Adaptive Server Support for Replication by Column Value* on page 73
- *sp_replication_path* on page 129
- *Changed Monitoring Tables* on page 135
- *monRepSchemaCache* on page 133

sp_dboption

Adaptive Server 15.7 SP100 introduces new parameters for the **sp_dboption** system procedure.

allow incremental dumps

The **allow incremental dumps** parameter supports back-up and restoration of cumulative dumps.

For example:

```
use master
go
sp_dboption mydb, "allow incremental dumps", true
go
```

If you do not first enable **allow incremental dumps**, you cannot perform a cumulative dump on your database.

sp_depends

Adaptive Server 15.7 SP100 introduces a new option in the **sp_depends** system procedure to display information about precomputed result sets.

sp_depends

sp_depends displays information about precomputed result sets. The syntax is:

```
sp_depends precomputed_result_set_name[, column_name]
```

Where:

- *precomputed_result_set_name* – is the name of the precomputed result set for which you are checking dependencies.
- *column_name* – is the name of the column on which you are checking dependencies.

sp_depends displays information about:

- Table and view dependencies in the database on which the specified precomputed result set depends
- Views in the database that depend on the precomputed result set
- Precomputed result set column dependencies and unique constraints defined in either the column specified, if you provide a *column_name*, or on all the columns in the precomputed result set, if you do not provide a *column_name*

The following examples assume that `prs1` and `view1` are created with the following dependency structure:

- `prs1` is defined on base table `tab1` (with unique constraint on column `c1`) and `view1` is defined on prs1
- `prs1` is configured for immediate refresh

This example displays the dependencies for table `tab1`, which includes a dependency on the `prs1` precomputed result set:

```
sp_depends tab1

Things inside the current database that reference the object.

Object                        type
--------                      ----------------------
dbo.prs1                      precomputed result set

The specified column (or all columns, if none was specified) in tab1
has no dependencies on other objects, and no other object depends on
any columns from it.
```

This example displays the precomputed result sets that include dependencies for column `c1`:

```
sp_depends prs1,c1
Things the object references in the current database.
object       type           updated    selected
------------ -------------- ---------- ---------
dbo.tab1     user table     no         no

Things inside the current database that reference the object.
object             type
------------------ --------
dbo.view1          view

Dependent objects that reference column c1.
Columns referenced in stored procedures, views or triggers are not
included in this report.
Type  Property   Object Names or Column Names Also see/Use
command
----- ---------- ---------------------------- --------------------
index constraint prs1_10240036482 (c1)        sp_helpindex,
                                              drop index,
                                              sp_helpconstraint,
                                              alter table drop
                                              constraint
```

sp_dump_history

In Adaptive Server 15.7 SP100, the **sp_dump_history** system procedure supports the same functionality for cumulative dumps as for full database dumps when you specify the **cumulative** keyword in its *@dump_type* parameter.

See also

- *Cumulative Dump* on page 13

sp_extrapwdchecks

sp_extrapwdchecks now allows NULL values for **caller_password** and **loginame** parameters.

The **caller_password** parameter is NULL when :

- The system security officer creates a new login account using **create login** command.
- The system security officer modifies the login account's password using **alter login … modify password** command.

The **loginame** parameter is NULL when:

- The system security officer creates a new login account using the **create login** command.

sp_helpdb

sp_helpdb output includes information about the durability of a user-created temporary database.

The **status** column of **sp_helpdb** includes these descriptions for database durability:

- `user created temp db` – normal temporary database created by the user (that is, created without specifying the **durability** parameter).
- `user-created enhanced performance temp db` – user-created temporary database created explicitly with the **no_durability** parameter. Because a database created with **no_durability** depends on licensing, it may not come online if the license expires.

For example, if you create this database:

```
create temporary database tempdb_explicit on default = 50
with durability = no_recovery
```

sp_helpdb includes this output:

```
sp_helpdb tempdb_explicit
 name db_size owner dbid created durability lobcomplvl inrowlen status
 ---- ------- ----- ---- ------- ---------- ---------- -------- ------

 tempdb_explicit  50.0 MB  sa  7    Dec 05, 2012 no_recovery  0  NULL
        select into/bulkcopy/pllsort, trunc log on chkpt,
        mixed log and data, user-created enhanced performance
        temp db, allow wide dol rows
```

```
(1 row affected)
device_fragments   size      usage         created          free  kbytes
----------------- ------ -------------- ------------------ -------------
 master           50.0 MB  data and log  Dec 5 2012 8:49PM    49216
(return status = 0)
```

Earlier versions of **sp_helpdb** did not indicate whether user-created temporary databases were explicitly created with a durability of **no_recovery**.

sp_helprotect

The **sp_helprotect** system procedure includes the owner name in the `object` column for objects (tables, databases, and so on).

sp_listener

sp_listener adds support for the AF_UNIX protocol.

The syntax is:

```
sp_listener "start", "afunix:host_name:pipe_name"
```

where:

- **afunix** – indicates you are adding an entry for an AF_UNIX connection.
- ***host_name*** – is the name of the local machine on which you start Adaptive Server. *host_name* and the local host name must match. You can use named pipes only for local communications.
- ***pipe_name*** – is the named pipe clients use to connect, in this format:

  ```
  /file_name/pipe_name
  ```

This example starts a connection on the machine named `big_server` and creates a pipe named `big_pipe` in the `/` (root) directory of this machine (Adaptive Server must be running on machine big_server):

```
 sp_listener "start", "afunix:big_server:/big_pipe"
```

sp_optgoal

Adaptive Server Enterprise version 15.7 SP100 changes permission checks for **sp_optgoal** based on your granular permissions settings.

When granular permissions is:

- Enabled – you must be a user with `manage opt goal` to create or delete a goal.
- Disabled – you must be a user with sa_role to create or delete a goal

Any user can run **sp_optgoal show**.

Once a goal is created, all users can use the goal.

See also

- *Permission Changes for Commands and Functions* on page 147

sp_replication_path

sp_replication_path supports binding, unbinding, and listing of filters.

The list action also now provides information about the default path, and indicates whether bindings are active based on the current distribution model. The object type has been extended to support replication filters using `filter` for the `bind`, `unbind`, and `list` actions. The filter type is shown as `RF` in all output.

For syntax, parameter descriptions, and examples, see *Replication Path Bindings* in the *Replication Server New Features Guide*.

See also

- *Adaptive Server Support for Replication by Column Value* on page 73
- *sp_config_rep_agent* on page 124
- *Changed Monitoring Tables* on page 135
- *monRepSchemaCache* on page 133

sp_showoptstats

Adaptive Server 15.7 SP100 introduces a new option in the **sp_showoptstats** system procedure to display information about precomputed result sets.

The syntax is:

```
sp_showoptstats
[[database_name.[owner].]{table_name|prs_name}],[column_name], [h]
```

where *prs_name* is the name of the precomputed result set for which you are displaying statistics.

sp_showoptstats displays additional information when you specify:

- No parameter – includes statistical information for all precomputed result sets in the current database.
- A precomputed result set – includes statistical information for precomputed result sets.

This example shows **sp_showopstats** output for the `prs1` precomputed result set:

```
sp_showoptstats prs1
-------------------------------------------------------------------------
 <?xml version="1.0" encoding="UTF-8"?>
 <optStats>
    <procVersion>sp_showoptstats/1.1/AnyPlatform/AnyOS/
                 Tues April 3 14:21:21 2012</procVersion>
    <serverVersion>Adaptive Server Enterprise/15.7.1/EBFXXXXX SMP
     ''/P/x86_64/Enterprise Linux/asecarina/ENG/64-bit/DEBUG/Mon
     Jul 9 00:16:37 2012</serverVersion>
    <serverName></serverName>
    <specifiedDatabase>prsdb</specifiedDatabase>
    <specifiedTableOwner></specifiedTableOwner>
    <specifiedTable>prs1</specifiedTable>
    <specifiedCol></specifiedCol>
    <tables>
        <tableOwner>dbo</tableOwner>
        <tableName>prs1</tableName>
        <tableType>precomputed result set</tableType>
        <tableStats>
. . .

        </noStatsCol>
    </tables>
</optStats>
```

sp_sjobcreate and sp_sjobmodify

sp_sjobcreate and **sp_sjobmodify** include the **continuous_run** parameter, which allows you to execute jobs from a starting time and have them run according to specified intervals until an end time.

The syntax for **sp_sjobcreate** is:

```
sp_sjobcreate @name='jsname', @options='server . . .
sproperties = continuous_run . . . '
```

The syntax for **sp_sjobmodify** is:

```
sp_sjobmodify @name='...', sproperties ='continuous_run'
```

where **continuous_run** indicates that you are scheduling a job that runs at the scheduled times during the specified interval.

This example uses **continuous_run** to schedule a job that runs every 10 minutes, starting at 2:00 p.m. on May 14, 2012 and ending on May 16 at 4:00 p.m.:

```
sp_sjobcreate 'sjname=sjob', @option='repeats=10minutes,
startdate=14 May 2012,enddate=16 May 2012,starttime=02:00pm,
endtime=04;00pm,sproperties=continuous_run’
```

This example create a job that runs every 10 minutes, starting at 2:00 p.m. on May 14, 2012 and ending on May 16 at 4:00 p.m.:

```
sp_sjobcreate 'sjname=sjob1', @option=’repeats=10minutes,
startdate=14 May 2012,enddate=16 May
2012,starttime=02:00pm,endtime=04;00pm'
```

Next, the above example uses the **continuous_run** property to configure the job to run every 10 minutes during that scheduled interval:

```
 sp_sjobmodify 'sjname=sjob1', @option='sproperties=continuous_run'
```

This example creates a job that executes every 2 hours (with a *days* constraint) between May 14, 2012 and May 20, 2012 on Monday, Wednesday, and Friday:

```
sp_sjobcreate 'sjname=sjob', @option='repeats=2hours,startdate=14
May 2012,enddate=20 May2012,starttime=02:00pm,endtime=06:00pm,
days=Monday:Wednesday:Friday,sproperties=continuous_run'
```

This example creates a job with a *dates* constraint that executes on May 1, 2012 at 2 p.m., 6 p.m., 10 p.m., and on May 3, 2012 at 2 a.m., 6 a.m., 10 a.m., 2 p.m., 6 p.m., and 10 p.m.:

```
sp_sjobcreate 'sjname=sjob', @option='repeats=4hours,startdate=1
May 2012,enddate=5 May
2012,starttime=02:00pm,endtime=07:00pm,dates=1:3,
sproperties=continuous_run'
```

This example runs the scheduled job that executes at 2:00 p.m. between May 14, 2012 and May 16, 2012:

```
sp_sjobcreate ‘sjname=sjob’, @option=’repeats=1day,startdate=14
May 2012,enddate=16 May 2012,starttime=02:00pm,endtime=04:00pm,
sproperties=continuous_run'
```

This example runs the scheduled job every 25 hours on repeating days, executing on May 14, 2012 at 2:00 p.m., on May 15, 2012 at 03:00 p.m., and on May 16, 2012 at 04:00 p.m.:

```
sp_sjobcreate 'sjname=sjob', @option='repeats=25hours,startdate=14
May 2012,enddate=16 May 2012,starttime=02:00pm,endtime=04:30pm,
sproperties=continuous_run'
```

Tables

Adaptive Server 15.7 SP100 includes new and changed tables.

New System Tables

New Adaptive Server 15.7 SP100 system tables.

sysdams

sysdams stores the dump allocation map (DAM) for the database. The DAM stores the list of allocation units that have been modified since the last full database dump. It is a bitmap with one bit per allocation unit in the database.

A value of:

- 0 – indicates that no page in the allocation unit has changed since the last full database dump.
- 1 – indicates that at least one page in the allocation unit has changed since the last database dump.

sysdams is automatically increased in size by an **alter database** operation. You cannot select from or view **sysdams**.

See also

- *Cumulative Dump* on page 13

Changed System Tables

Adaptive Server 15.7 SP100 includes changes to existing system tables.

sysloginroles

`sysloginroles` adds the `predid` column:

Name	Datatype	Description
`predid`	`int`	The object ID for the predicate of a **grant role** command. See *Predicated role activation* in *Chapter 7: Granting Predicated Privileges* of the *Security Administration Guide*.

New Monitoring Tables

Adaptive Server 15.7 SP100 adds new monitoring tables.

The new monitoring tables are:

- `monRepCoordinator` – provides information on the RAT coordinator process when RAT runs in Multiple Scanner mode.
- `monRepSchemaCache` – provides information about the schema cache for each scanner running on a Replication Server.
- `monSysExecutionTime` – includes one row for each operation module executed by Adaptive Server.

monSysExecutionTime

The `monSysExecutionTime` monitoring table includes one row for each operation module executed by Adaptive Server.

Enable the **enable monitoring** and **execution time monitoring** configuration parameters for this monitoring table to collect data.

Columns

The columns for `monSysExecutionTime` are:

Name	Datatype	Attribute	Description
`InstanceID`	`smallint`		(Cluster environments only) ID of an instance in a shared-disk cluster
`OperationID`	`int`		Unique ID of an operation category
`OperationName`	`varchar(30)`		Name of the operation category

Name	Datatype	Attribute	Description
`ExecutionTime`	`bigint`	counter	Execution time, in microseconds, of each operation performed
`ExecutionCnt`	`bigint`	counter	Total number of occurrences of this operation type

monRepCoordinator

`monRepCoordinator` provides information on the RAT coordinator process when RAT runs in Multiple Scanner mode.

Besides providing general information about the coordinator process, such as its spid, or the database with which RAT is associated, it also provides status information.

Columns

The columns for `monRepCoordinator` are:

Name	Datatype	Description
`DBID`	`smallint`	Unique identifier for the database currently being used by the process
`SPID`	`smallint`	Coordinator process identifier
`InstanceID`	`tinyint`	(Cluster environments only) ID of an instance in a shared-disk cluster
`DBName`	`varchar(30)`	Database name for this Rep Agent
`Status`	`varchar(30)`	Current task status
`SleepStatus`	`varchar(30)`	Current sleep status, if sleeping

monRepSchemaCache

The `monRepSchemaCache` table reports the schema cache information for each scanner running on a Replication Server.

Columns

The columns for *monRepSchemaCache* are:

Description	Datatype	Description
`DBID`	`int`	Unique identifier for the database running the Replication Agent.

Description	Datatype	Description
`ScannerSpid`	`int`	Session process identifier of the scanner task.
`InstanceID`	`tinyint`	(Cluster environments only) ID of an instance in a shared-disk cluster.
`ConfiguredSize`	`int`	Size of the schema cache, in bytes, as configured using `max schema cache per scanner`.
`CurrentUsageSize`	`int`	Current size, in bytes, of the schema cache for this scanner.
`MaxReachedSize`	`int`	Maximum size, in bytes, reached for the schema cache.
`ObjectSchemas`	`int`	Number of schemas in cache for tables/stored procedures.
`TextImageDescriptors`	`int`	Number of descriptors for text/ image column replication.
`WideParameters`	`int`	Number of descriptors for the wide parameter for stored procedure replication.
`ObjectSchemasFlushed`	`int`	Number of table/stored procedure schemas that have been flushed.
`TexImageDescriptorsFlushed`	`int`	Number of descriptors for text/ image column replication that have been flushed.
`WideParametersFlushed`	`int`	Number of descriptors for the wide parameter for stored procedure replication that have been flushed.
`CacheTooSmallFlushes`	`int`	Number of objects flushed because the schema cache could not hold all schemas involved. This might be an indication that you need to increase the schema cache size.

Description	Datatype	Description
`TotalAllocTime`	`int`	Total amount of time spent allocating objects. This counter is updated only when Adaptive Server monitoring is enabled.
`TotalDeallocTime`	`int`	Total amount of time spent de-allocating objects. This counter is updated only when Adaptive Server monitoring is enabled.
`DBName`	`varchar(30)`	Name of the database in which the task scans.

See also

- *Adaptive Server Support for Replication by Column Value* on page 73
- *sp_config_rep_agent* on page 124
- *sp_replication_path* on page 129
- *Changed Monitoring Tables* on page 135

Changed Monitoring Tables

Adaptive Server 15.7 SP100 includes changes to existing monitoring tables.

monOpenObjectActivity

monOpenObjectActivity adds these colums:

Description	Datatype	Description
`NumLevel0Waiters`	`float`	Number of times a Level0 Scan start waited because of a utility's wait request.
`AvgLevel0WaitTime`	`float`	Average time, in milliseconds, Adaptive Server waited for Level0 access.

monRepLogActivity

The `MaxHashSchemaSize` and `NumberOfSchemasReused` columns of the `monRepLogActivity` table have been removed.

monRepScanners

The `NumberOfTruncPointRequested` and `NumberOfTruncPointMoved` columns are populated only when the single task scanning of a log model is used. When the Multiple Replication Paths (MRP) model is used, these columns values are 0. The `monRepSenders` table contains related information for the MRP model.

monRepSenders

Description	Datatype	Description
`NumberOfTruncPointRequested`	`int`	Total number of times RepAgent asked Replication Server for a new truncation point.
`NumberOfTruncPointMoved`	`int`	Total number of times RepAgent moved the secondary truncation point.
`AvgTruncPointInterval`	`int`	Displays the average time between truncation point requests. This value is an indication of the effectiveness of the `trunc point request interval` configuration parameter. If `trunc point request interval` is set correctly, the values of `trunc point request interval` and `AvgTruncPointInterval` will be the same.

monSpinlockActivity

Name	Datatype	Attribute	Description
`SpinlockSlotID`	`int`		ID for this spinlock in the spinlock memory pool

monState

Enable the `enable monitoring` configuration parameter for this monitoring table to collect data.

Name	Datatype	Attribute	Description
`TableAccesses`	`bigint`	Counter	Number of pages from which data was retrieved without an index
`IndexAccesses`	`bigint`	Counter	Number of pages from which data was retrieved using an index
`ULCFlushes`	`bigint`	Counter	Total number of times the User Log Cache was flushed

Name	Datatype	Attribute	Description
`ULCFlushFull`	`bigint`	Counter	Number of times the User Log Cache was flushed because it was full
`WorkTables`	`bigint`	Counter	Total number of work tables created
`TempDBObjects`	`bigint`	Counter	Total number of temporary tables created
`Rollbacks`	`bigint`	Counter	Total number of transactions rolled back
`Selects`	`bigint`	Counter	Total number of **select** operations executed
`Updates`	`bigint`	Counter	Total number of **update** operations executed
`Inserts`	`bigint`	Counter	Total number of **insert** operations executed
`Deletes`	`bigint`	Counter	Total number of **delete** operations executed
`Merges`	`bigint`	Counter	Total number of **merge** operations executed
`ULCKBWritten`	`bigint`	Counter	Number of kilobytes written to the user log cache
`PagesRead`	`bigint`	Counter	Number of pages read server-wide
`PagesWrite`	`bigint`	Counter	Number of pages written server-wide
`PhysicalReads`	`bigint`	Counter	Number of buffers read from the disk
`PhysicalWrites`	`bigint`	Counter	Number of buffers written to the disk
`LogicalReads`	`bigint`	Counter	Number of buffers read from cache

See also

- *Adaptive Server Support for Multiple Scanner Threads* on page 71

Utilities

Adaptive Server 15.7 SP100 includes new and changed utilities.

New Utilities

New Adaptive Server 15.7 SP100 utilities.

sybdumptran

Use the **sybdumptran** utility to dump the most recent transactions when the database and the server have suffered a catastrophic failure.

sybdumptran generates a transaction log dump from the log pages contained in operating system files/raw devices, that were formerly used as a log device by a database in an Adaptive Server environment. **sybdumptran** is located in:

- (UNIX) `$SYBASE/$SYBASE_ASE/bin/`
- (Windows) `%SYBASE%\%SYBASE_ASE%\bin` as `sybdumptran.exe`

Syntax

```
sybdumptran [-m file_name | -g | -d database_name |
    -f first_page_num ] -o output_file | -h dump_history_file_name
```

or:

```
sybdumptran --help
```

Parameters

- ***-m file_name*** – specifies the location of the metadata file containing information needed by **sybdumptran** to dump the transaction log, including the path to the database log devices, and the layout of the log segment relative to these devices. It also contains sequencing information and the page number of the first page in the log. This metadata file can be:
  - A file generated using the `-g` option; or,
  - Preferably, the most recent full database dump or transaction log dump.

    Use the most recent database dump/transaction log to ensure that the information is current, including the location of the log segment relative to the rest of the database, the dump sequencing information, and the location of the active log within the log segment. If it is not possible to use the most recent backup, then use a backup that was taken when the database had the same layout as it currently has. This means that the sequencing information and log location will be incorrect. In this case, the **sybdumptran** utility automatically tries to determine the location of the active log, and the sequencing must be overridden when the transaction log is loaded using the `dump tran with override=sequence` command.

  The long parameter option displays when you use `--help`, so you can use either `-m` *`filename`* or `--metadata-file=`*`filename`*.
- **`-g`** – generates a compact metadata file from a database or transaction log dump. Specify the name of the output file using the `-o` *`output_file`* option. You can use the `-g` option only with the `-m` and `-o` options.

  If the most recent database or transaction log dump is unavailable, or when you use `-g` to generate a metadata file, you can use an older database or transaction log dump as the information source for providing the layout of the database if it was identical at the time the dump was performed, to compare with the metadata file you just generated. The dump from which it is generated, however, should be the most recent.

The `-g` option to generate metadata is useful when the last database or transaction log dump is located on an entirely different system. The created metadata file is a subset of the database dump or transaction log dump from which it was created, containing only the information needed by **sybdumptran**. Because it is smaller, it is easier to copy to the system on which you are creating the transaction log.

- **`-d`** ***`database_name`*** – is the name of the database used to locate and add entries to the dump history file.
- **`-h`** ***`dump_history_file_name`*** – used in combination with the `-d` option, adds an entry to the dump history file for the transaction log dump generated by **sybdumptran**. When you use `-h` *`dump_history_file_name`*, **sybdumptran** tries to locate the most recent dump that was taken, and uses this as its source of the metadata, obviating the need for the `-m` option.

  The entry is generated for the database named by the `-d` option. No entry is added in the dump history file if this database has no existing entries.

  Note: You cannot use `-h` with the `-m` *`file_name`* option.
- **`-f`** ***`first_page_num`*** – allows you to specify the first log page.

  The **sybdumptran -m** option uses the metadata file to locate the first page of the log within the log devices. If the metadata file is not the most recent transaction log or database dump, then the first page of the log is incorrect, and **sybdumptran** automatically tries to locate the first page of the log. If this fails, and **sybdumptran** exits and reports that it cannot locate the first log page, use the `-f` option to manually specify the first log page.
- **`sybdumptran --help`** – displays this long parameter format:

```
Usage: sybdumptran <option list>
Valid options:
  -d, --databasename=database_name      - name of database in dump
                                          history
  -h, --dump history file=filename      - dump history file name
  -f, --first-log-page=page_number      - to overwrite the first
                                          page from meta data
  -g, --generatemetadata                - create compact meta
                                          data file from dump
  -H, --help=[{0|1|2|3}[,display_width]] - print this help
                                          message, and exit
  -m, --metadatafile=filename           - meta data file used
                                          to locate devices
  -o, --outputfile=filename             - name of output file
  -V, --sbssav                          - print a short version
                                          string, and exit
  -T, --trace=number                    - for debugging
  -v, --version                         - print version message,
                                          and exit
```

Examples

- **sybdumptran Example –** Start with a database dump (`/dumps/db.dmp`), and two transaction logs (`/dumps/db.trn1` and `/dumps/db.trn2`).

  Use the last transaction log dump as a metadata file (`–m` for **sybdumptran**):

  ```
  > sybdumptran -o /dumps/db.trn_sdt -m /dumps/db.trn2
  Opening output-file '/dumps/db.trn_sdt'.
  Opening metadata-file '/dumps/db.trn2'.
  Opening log devices used by database:
      Opening device "db_log1", path "/sdc1_eng/devices/db.log1".
      Opening device "db_log2", path "/sdc1_eng/devices/db.log2".
  Building run-lists using first log page=10888.
  Finished building run-lists, number of log pages=7, first log
  page=10888,
      last log page=10894.
  Dumping log pages to output file "/dumps/db.trn_sdt".
  Finished dumping 7 log pages.
  Sybdumptran completed without errors.
  ```

  Load the database dump, the two transaction log dumps, and the dump generated by **sybdumptran**:

  ```
  1> load database db from '/dumps/db.dmp'
  2> go
  1> load tran db from '/dumps/db.trn1'
  2> go
  1> load tran db from '/dumps/db.trn2'
  2> go
  1> load tran db from '/dumps/db.trn_sdt'
  2> go
  1> online database db
  2> go
  ```

  When you do not use the most recent dump as metadata file, the dump generated by **sybdumptran** contain the wrong sequence date. This example uses `/dumps/db.trn1` as the metadata file:

  ```
  > sybdumptran -o /dumps/db.trn_sdt -m /dumps/db.trn1
  Opening output-file '/dumps/db.trn_sdt'.
  Opening metadata-file '/dumps/db.trn1'.
  Opening log devices used by database:
      Opening device "db_log1", path "/sdc1_eng/devices/db.log1".
      Opening device "db_log2", path "/sdc1_eng/devices/db.log2".
  Building run-lists using first log page=10253.
  Found new first log page=10888.
  Restarting the building of run-lists.
  Building run-lists using first log page=10888.
  Finished building run-lists, number of log pages=7, first log
  page=10888,
      last log page=10894.
  Dumping log pages to output file "/dumps/db.trn_sdt".
  Finished dumping 7 log pages.
  Sybdumptran completed without errors.

  1> load database db from '/dumps/db.dmp'
  ```

```
2> go
1> load tran db from '/dumps/db.trn1'
2> go
1> load tran db from '/dumps/db.trn2'
2> go
1> load tran db from '/dumps/db.trn_sdt'
2> go
Backup Server session id is: 69. Use this value when executing the
'sp_volchanged'
system stored procedure after
 ..
Msg 4305, Level 16, State 1:
Server 'marslinux1_asecarina_smp', Line 1:
Specified file 'dump device' is out of sequence. Current time
stamp is Nov 30 2012
1:59:59:423AM while dump was from Nov 30 2012  1:59:59:296AM.
```

To be able to load this dump, use the `override` option:

```
1> load tran db from '/dumps/db.trn_sdt'
2> with override = sequence
3> go
Backup Server session id is: 83. 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 'db1233501C1F
' section number 1 mounted on disk file '/dumps/db.trn_sdt'
Backup Server: 4.58.1.1: Database db: 22 kilobytes LOADED.
...
1> online database db
2> go
Started estimating recovery log boundaries for database 'db'.
...
```

Enable the dump history file:

```
1> sp_configure 'dump history filename', 1, '/dumps/dump_hist'
2> go
1> sp_configure 'enable dump history', 1
2> go
```

Dump database is then added to the history file:

```
1> dump database db to '/dumps/db.dmp'
2> go
```

Permissions

Any user who has read permissions on the log devices of the database, and write permission to the output file can use **sybdumptran**.

Options for load transaction

During a load sequence of a database dump and one or more transaction log dumps, Adaptive Server verifies that the transaction log dump is loaded in sequence based on a sequence number stored in the dump header.

If you do not use the most recent dump as a metadata file for a transaction log dump generated by **sybdumptran**, the sequence number will be incorrect, causing the load of this transaction log dump to fail. The **load transaction** option **with override=sequence** ignores this error and allows the load of an out-of-sequence transaction log dump.

If the sequence number **sybdumptran** created for the dump does not match the new sequence number of the previous dump, restore the database using the **load tran** command with the **with override=sequence** option. You cannot use this option when loading transaction dumps created by Adaptive Server, but only when loading transaction logs created with **sybdumptran**.

Warning!

- When using the **override=sequence** option, make sure the transaction log to be loaded is indeed the next in the sequence. Since the sequence number is there to protect the sequence, overriding it means the task must be performed correctly. Loading a transaction log out of sequence can corrupt the database (for example, if an earlier dump transaction is omitted during the load, so that the load sequence would need to be repeated, this time, in the correct order).
- If a partially logged operation (such as **select into** without the **enable full logging for select into** database option) was done between the last transaction log that was dumped normally, and the transaction log that is created by **sybdumptran**, then a sequence error will not be reported during the load of the latter, even when using the most recent transaction log dump as a metadata file. Normally, a partially logged operation deliberately breaks the sequence to make sure that a subsequent dump transaction does not take place.

  In such a case, do not load the dump from **sybdumptran**. **sybdumptran** dumps the transaction log, irrespective of whether partially logged operations have taken place or not, because the utility does not have access to the metadata information held within the database. If the sequencing is overridden when the transaction log is loaded, the load may fail, or worse, the database may become corrupted.

  If you make use of partially logged operations and you are unsure whether such an operation took place before the **sybdumptran** operation, make sure to run database consistency checks after the transaction log created by **sybdumptran**, has been loaded.

In this example, when you omit the **-m** metadata file option and specify the dump history file, **sybdumptran** locates the metadata file in the dump history file:

```
> sybdumptran -o /dumps/db.trn_sdt -h /dumps/dump_hist -d db
Opening output-file '/dumps/db.trn_sdt'.
Opening dump-history-file '/dumps/dump_hist'.
Option 'metadata-file' is not supplied, using last dump '/dumps/
```

```
db.trn1' from dump history file.
Opening metadata-file '/dumps/db.trn1'.
Opening log devices used by database:
    Opening device "db_log1", path "/sdc1_eng/devices/db.log1".
    Opening device "db_log2", path "/sdc1_eng/devices/db.log2".
Building run-lists using first log page=10894.
Finished building run-lists, number of log pages=1, first log
page=10894, last log page=10894.
Dumping log pages to output file "/dumps/db.trn_sdt".
Finished dumping 1 log pages.
Sybdumptran completed without errors.
```

Changed Utilities

Changed Adaptive Server 15.7 SP100 utilities.

ddlgen

An X*extended_object_type*, –XDE has been added for -TDB. The new type allows you to generate a database and all of its objects in correct dependent order. For example, you can issue the following to generate the DDL in the order listed below:

```
ddlgen –S –U –P –TDB –Ndbname –XDE
```

1. Segment
2. Group
3. User
4. Rules
5. Defaults
6. UDDs
7. Encrypted Keys
8. User Tables
9. Proxy Tables
10. Triggers
11. Functions and Views
    a. all functions without any dependency
    b. all views without any dependency
    c. all functions and all views with any dependency on any functions and views
12. Instead of trigger
13. Stored Procedures
14. Extended Stored Procedures
15. PRS
16. User Defined Web Services

optdiag

optdiag adds the *prs_name* parameter. The changed syntax is:

```
optdiag [binary] [simulate] statistics
    { -i input_file | database[.owner[.[{table|prs_name}
[.column] ] ] ] [-o output_file] }
    . . .
```

where *prs_name* indicates the name of the precomputed result set.

When you specify:

- A database name – **optdiag** includes information about statistics for any precomputed result sets in the database.
- Precomputed result set – **optdiag** includes statistical information for precomputed result sets.

This example displays information for the precomputed result set `prs1`:

```
optdiag statistics prsdb..prs1 -Usa -Ppass –Sserver

Server name:                            ""

Specified database:                     "prsdb"
Specified table owner:                  not specified
Specified table:                        "prs1"
Specified column:                       not specified

Table owner:                            "dbo"
Table name:                             "prs1"
Table type:                             "precomputed result set"

Statistics for table:                   "prs1"

. . .

No statistics for remaining columns:    "a"
(default values used)

Optdiag succeeded.
```

sybrestore

The **sybrestore** utility adds support for password protect dump files. When restoring a database, you must provide the password for dump files that are password protected.

Configuration Parameters

Adaptive Server 15.7 SP100 introduces new and changed configuration parameters

Changed Configuration Parameters

Adaptive Server 15.7 SP100 changes for configuration parameter `enable permissive unicode`.

enable permissive unicode

Adaptive Server versions 15.7 SP100 and later changes the `enable permissive unicode` configuration parameter, which allows you to include random binary data when enabled (set to 1). However, once you enable `enable permissive unicode`, Adaptive Server correctly sorts only valid UTF-8 data.

permission cache entries

Adaptive Server version 15.7 ESD #2 and later have a default value of 64 for the **permission cache entries** configuration parameter.

New Configuration Parameters

Adaptive Server 15.7 SP100 includes new configuration parameters.

optimize temp table resolution

Table 3. Summary Information

Default value	0
Range of values	0, 1
Status	Static
Display level	Comprehensive
Required role	System Administrator
Configuration group	Query Tuning

Allows stored procedures that reference temporary tables created outside the procedure to not require recompiling for each execution. Trace flag 299 previously supported this functionality.

enable ISM

Table 4. Summary Information

Default value	0
Range of values	0 – 2
Status	Static
Display level	
Required role	System Administrator
Configuration group	Physical Memory

Enables and disables Adaptive Server to use integrated service management (ISM) on the Solaris platform. Valid values are:

- 0 – use ISM if possible; if ISM is not available, use regular shared memory.
- 1 – use only regular shared memory.
- 2 – use only ISM.

enable HugePages

Table 5. Summary Information

Default value	0
Range of values	0 – 2
Status	Static
Display level	
Required role	System Administrator
Configuration group	Physical Memory

Enables and disables Adaptive Server to use huge pages on Linux platforms that support huge pages. Valid values are:

- 0 – use huge pages if possible; if huge pages are not available, use regular pages.
- 1 – use only regular pages.
- 2 – use only huge pages.

max util parallel degree

Table 6. Summary Information

Default value	1
Range of values	1– 255
Status	Dynamic
Display level	Basic
Required role	System Administrator
Configuration group	Query Tuning

max util parallel degree specifies the server-wide maximum number of worker processes allowed per query used by the **create index with consumers** and **update stats with consumers** commands.

solaris async i/o mode

Table 7. Summary Information

Default value	0
Values	0, 1
Status	Static
Display level	
Required role	System Administrator
Configuration group	Disk I/O

Allows you to select various asynchronous IO modes on the Solaris platform. This parameter is effective if Adaptive Server is running in threaded kernel mode. This parameter is static, therefore is effective after restarting Adaptive Server.

0 – (Default) Use this mode if the Solaris patch containing the fix for Oracle BugID 16054425 is not installed. You may see sub-optimal IO performance.

1 – (Recommended) You must have the Solaris patch containing the fix for Oracle BugID 16054425 installed.

Install the following Oracle patch for your platform:

- For Solaris 10 SPARC: 148888-03
- For Solaris 10 x86/x64: 148889-03
- For Solaris 11, latest SRU containing fix for Oracle Bug 16054425

Note: If **solaris async i/o mode** is set to 1 without the patch for Oracle BugID 16054425, Adaptive Server may report 694 or 823 errors and require restarting the server. Oracle Bug 15868517 refers to backport of Oracle Bug 16054425 for S10U11.

See also

- *Solaris Asynchronous I/O Performance* on page 101

Permission Changes for Commands and Functions

Adaptive Server 15.7 SP100 includes permission changes for commands and functions.

*Changes to **asehostname** Permissions*

In versions of Adaptive Server earlier than Adaptive Server 15.7 SP100, only users with the sa_role could execute **asehostname**. For Adaptive Server 15.7 SP100 and later, **asehostname** has these permission requirements:

With granular permissions enabled, to query the hostname on which the dataserver is running, you must have `manage server` privileges, or been granted `select` permission on **asehostname**.

With granular permissions disabled, you must be a user with sa_role or have `select` permission on **asehostname** in order to query the hostname on which the dataserver is running.

Changes to dbcc Commands Permissions

- **dbcc pravailabletempdbs** - With granular permissions enabled, you must be a user with `manage server` privilege to execute **pravailabletempdbs**. With granular permissions disabled, you must be a user with sa_role.
- **dbcc serverlimits** - With granular permissions enabled, you must be a user with `manage server` privilege to execute **serverlimits**. With granular permissions disabled, you must be a user with sa_role.
- **dbcc cis showcaps** - With granular permissions enabled, you must be a user with `manage server` privilege to execute **cis showcaps**. With granular permissions disabled, you must be a user with sa_role.
- **dbcc cis remcon** - With granular permissions enabled, you must be a user with `manage server` privilege to execute **cis remcon**. With granular permissions disabled, you must be a user with sa_role.

Changes to sp_optgoal Permissions

A new server level privilege, `manage opt goal` has been created to enable users with sa_role and sa_serverprivs_role to write or delete goals using **sp_optgoal**.

With granular permissions enabled, you must be a user with `manage opt goal` privilege in order to write or delete a goal. By default, sa_role and sa_serverprivs_role are granted the `manage opt goal` privilege.

With granular permissions disabled, you must be a user with sa_role in order to write or delete a goal.

Any user can run `sp_optgoal 'show'.`

See also

- *sp_optgoal* on page 128

Index