SYBASE®

An SAP® Company

New Features Guide

# Adaptive Server® Enterprise

# 15.7 SP110

# Contents

# Improved Data-Load Performance

Adaptive Server® improves the **ins_by_bulk** optimizer criteria, providing faster data loading by using parallel index updates for insert statements into a datarows-locked table with non-clustered indexes.

Enabling the data-load optimization criteria with **set ins_by_bulk** runs subsequent **insert...select** statements using bulk insert methods.

In this update, Adaptive Server provides enhancements to improve the performance of data loads via the **insert** statement to datarows-locked tables with two or more non-clustered indexes. Index updates are done in parallel to speed up the performance of the data load.

## Data-Load Optimization on Tables with Multiple Indexes

Data-load optimization on a target table that has multiple indexes efficiently loads data pages and maintains multiple indexes in parallel.

To perform maintenance on multiple indexes in parallel, you must configure Adaptive Server with resources (for example, threads), by setting:

- **number of worker processes** to the maximum that Adaptive Server can use at any one time for all simultaneously running queries.
- **max parallel degree** to the maximum for a query. This parameter determines the maximum number of threads Adaptive Server uses when processing a given query.

See *Query Processing and Abstract Plans* in *Performance and Tuning Series*.

### Viewing Parallel Update Index Queries in showplan

**set showplan** uses the **update index** operator to display parallel update index statements.

For example:

```
set showplan on
set ins_by_bulk on
go

insert into my_authors select * from authors
```

This example showplan output is for an insert-select into a table with several indexes, executed with the data load optimizations. The text shown in **bold** reports the parallel index update enhancements.

```
QUERY PLAN FOR STATEMENT 1 (at line 2).
Optimized using Parallel Mode
Executed in parallel by coordinating process and 2 worker processes.
```

```
 STEP 1
     The type of query is INSERT.

     5 operator(s) under root

    |ROOT:EMIT Operator (VA = 5)
    |
    |    | INSERT Operator (VA = 4)
    |    |  The update mode is direct.
    |    |
    |    |   |SCAN Operator (VA = 0)
    |    |   |  FROM TABLE
    |    |   |  authors
    |    |   |  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.
    |    |
    |    |   |EXCHANGE Operator (VA = 3) (Merged)
    |    |   |Executed in parallel by 2 Producer and 1 Consumer processes.

    |    |   |
    |    |   |    |EXCHANGE:EMIT Operator (VA = 2)
    |    |   |    |
    |    |   |    |    |UPDATE INDEX Operator (VA = 1)
    |    |
    |    |  TO TABLE
    |    |  my_authors
    |    |  Using I/O Size 16 Kbytes for data pages.


The INSERT OPERATOR (VA = 4) is using BULK INSERT

(23 rows affected)
```

Without these data load enhancements, the data insertion is done in serial mode, with all the indexes being updated sequentially after the insertion of each row. With the enhanced data load optimizations, data pages are populated in bulk mode by one insert thread, followed by index updates which are performed by multiple threads each updating one or more indexes for all the data rows inserted on one page.

The line:

```
The INSERT OPERATOR (VA = 4) is using BULK INSERT
```

shows that bulk inserts are being performed.

The plan fragment **`UPDATE INDEX Operator (VA = 1)`** shows that the index updates are being performed using parallel worker threads.

## Viewing Queries with UpdateIndex

**show_cached_plan_in_xml** displays the cached execution plan using **UpdateIndex** to indicate that multiple indexes are applied in parallel.

This example shows the **UpdateIndex** operator output that updates multiple indexes in parallel:

```
<EmitXchg>
<Details>
<VA>2</VA>
   <Vtuple Label="OutputVtuple">
   <Collection Label="Columns(#1)">
   <Column>
   (0x0x1497d6f00) type:INT1len:1 offset:0 valuebuf:0x(nil)
status:(0x00000000)
(constant:0x0x14a10e760 type:INT1 len:1 maxlen:1 )
   </Column>
   </Collection>
   </Vtuple>
</Details>
   <UpdateIndex Label="UpdateIndex:">
   <Details>
      <VA>1</VA>
   </Details>
   </UpdateIndex>
</EmitXchg>
```

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

## Restrictions

Data load using parallel index updates optimization is unavailable in certain scenarios.

You cannot use data-load optimization:

- When no valid index is defined or only one index is in the target table.
- The target table is a temporary table or a worktable.

The following behaviour and restrictions are imposed when bulk insert is used in a transaction:

- Serial or concurrent data that is loaded using bulk inserts in a multistatement transaction is visible only to the inserting task while the transaction is active. While the transaction that loaded this data using bulk inserts remains active, concurrent access to this data by other tasks skips the uncommitted data.
- A single transaction can perform multiple bulk inserts to the same table multiple times. However, in a single transaction (whether it spans multiple databases, or only one), you can insert into a maximum number of four tables per database 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.

- Bulk insert is disabled while the following commands are underway concurrently when the data load is attempted. The insert instead performs in non-bulk mode, with serial index updates.
  - **reorg rebuild ... with online**
  - **create index ... with online**
- 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.

## Downgrade Considerations

Transaction logs generated by the data-load optimizations and parallel index updates are incompatible with versions of Adaptive Server earlier than 15.7 SP110. Database and transaction log dumps that are generated with data-load optimizations and parallel index updates cannot be recovered by earlier versions.

To downgrade an entire Adaptive Server installation to an earlier version, use **sp_downgrade**, which truncates the active portion of the transaction log, thus permitting recovery of the databases by the earlier version.

See the *Adaptive Server Enterprise 15.7 ESD #2 Reference Manual: Procedures*.

To load a database containing transactional activity from the optimized data load, use **sp_downgrade_esd** to first downgrade it to the target Adaptive Server support package version. This process clears transactional activity from the log, which then allows an earlier version of Adaptive Server to load and recover a dump of this database.

See the *Adaptive Server Enterprise 15.7 SP100 Installation Guide*.

You cannot downgrade database dumps that contain log activity from the optimized data load and parallel index updates. Use the output from **load...with headeronly** to see if a particular dump contains such activity. To load such dumps into an earlier Adaptive Server version that does not support these new optimizations (for example, Adaptive Server 15.7 SP100), you must first reload the dump into an Adaptive Server version that supports this feature. Then use **sp_downgrade_esd** to downgrade the database to the appropriate version. Re-create the dump from the downgraded database in a form that is compatible with the earlier Adaptive Server version.

With some restrictions, database and transaction log dumps from a later version of Adaptive Server can be loaded and recovered in earlier versions. However, once a dump is generated, it may be compatible with only a few of the earlier versions of Adaptive Server, depending on the features used in the database, or the objects that are contained in the dumps.

Prior to generating the dumps, you can use the **db_attr** system function to identify the features that are currently active in the database, and the information that will be included in dumps.

## Troubleshoot

Troubleshoot common situations that may cause bulk insert optimizations to not be applied.

- Check the showplan output to confirm that:
  - The `UPDATE INDEX` operator is used.
  - The `INSERT OPERATOR` is using `BULK INSERT`. It is possible at runtime Adaptive Server decides to run the query without `BULK INSERT` even though it was used during compile time.
  - A sufficient number of worker threads is configured or available to perform the index updates in parallel.
- If bulk insert does not execute, set trace flag 9586 to diagnose the reason in the XML output.

# Remote Dump Host Control

Backup Server introduces a remote access control feature that prevents remote dumps and loads, and execution of remote procedure calls (RPCs) from any client or server that is running on unauthorized servers.

### *Access Control File*

Authorization to dump to, or load from, Backup Server is achieved by including the authorized hosts in the `hosts.allow` file. The default name of the file is `hosts.allow`, which is by default located in `$SYBASE`. You can change the location and file name using:

`backupserver -h` *full_path_name*`/hosts.allow`

When you start Backup Server, the location of the file is shown in the error log. For example:

```
Backup Server: 1.88.1.1: The hosts authentication file used by
the backup server is '/remote/myServer/ase157x/SMP/release/
hosts.allow'.
```

If you do not specify a file, `$SYBASE/hosts.allow` is used. If the location of the file is a relative path, the path is replaced by the absolute path using the directory in which the Backup Server has been started. For example, if you start Backup Server from `/usr/u/myServer` and Backup Server is started with:

```
backupserver -h./myhosts.allow
```

The error log shows:

```
Backup Server: 1.88.1.1: The hosts authentication file used by
the backup server is '/usr/u/myServer/./myhosts.allow'.
```

If the file `hosts.allow` does not exist, dump or load commands, or remote procedures, fail.

**Note:** Local dumps are not affected by this feature.

### *File Content*

The format for `hosts.allow` is:

```
host_name_running_backupserver [ \t*][,][ \t*] host_name_trying_to_connect
```

```
host_name_running_backupserver:
hostname | hostname.domain | ipv4 address | ipv6 address
```

```
host_name_trying_to_connect:
hostname | hostname.domain | ipv4 address | ipv6 address |+

The '+' sign can be used as a wildcard to allow remote
dumps to, or loads from, any Backup Server running on
the specified host.
```

```
Example:
# Example of hosts.allow file
# Development machine imetsol1 allows access from everywhere
imetsol1 +

# Group development machine marslinuxX allow access from other
# marslinuxX machines only
marslinux1 marslinux2
marslinux1 marslinux3
marslinux2 marslinux1
marslinux2 marslinux3
marslinux3 marslinux1
marslinux3 marslinux2
```

### *Permissions*

The recommended file permission for UNIX is no greater than `640`. For Windows, ensure that only authorized users are granted access to the file.

### *Error and Warning Messages*

- On UNIX, if permission levels are set lower than `640`, you see a warning similar to:

  ```
  Backup Server: 1.86.1.1: Warning: The file './hosts.allow'
  has an unsafe permission mask 0664. The recommended value is
  0640.
  ```
- On Windows, if you have not established permissions, or if access is granted to everyone, you see a warning similar to:

  ```
  Backup Server: 1.87.1.1: Warning: The file './hosts.allow'
  either has no access control or one of the entries allows
  access to everyone. It is recommended that only the owner
  has permission to access the file.
  ```
- If you attempt to load to, or dump from, a remote Backup Server that does not have the appropriate access control record, you see error:

  ```
  Backup Server: 5.16.2.2: Client-Library error: Error number
  44, Layer 4, Origin 1, Severity 4: ct_connect(): protocol
  specific layer: external error: The attempt to connect to
  the server failed. Backup Server: 5.3.2.1: Cannot open a
  connection to the slave site 'REMOTE_BS'. Start the remote
  Backup Server if it is not running.
  ```
- If you attempt to execute an RPC on a remote Backup Server that does not have the appropriate access control record, you see error:

  ```
  Msg 7221, Level 14, State 2:
  Server 's', Line 1:
  Login to site 'REMOTE_BS' failed.
  ```

# Transaction Log Space Management

Adaptive Server introduces new functionality to analyze and free transaction log space.

Adaptive Server provides a single transaction log segment per database. Space can be added to a log segment or removed from a log segment. However, at any given point, there is limited space in a log segment.

Whenever the database client applications perform any data manipulation language (DML) operations on the data, Adaptive Server produces log records that consume space in the transaction log. Typically there are several clients performing DMLs concurrently on the database and log records are appended to the log whenever private local caches (PLCs) for individual clients are full or in some other conditions such as when a data page is changed by multiple open transactions. Log records of several transactions are therefore typically interleaved in the transaction log space.

Removing transactions from the transaction log to free the log space can be done using **dump transaction**. However, there different scenarios that can cause the transaction log can grow in such a way that the **dump transaction** command is not able to free space. In these situations, the log consumes space to such an extent that it affects the continuous availability of the database system for any write operations. The scenarios include:

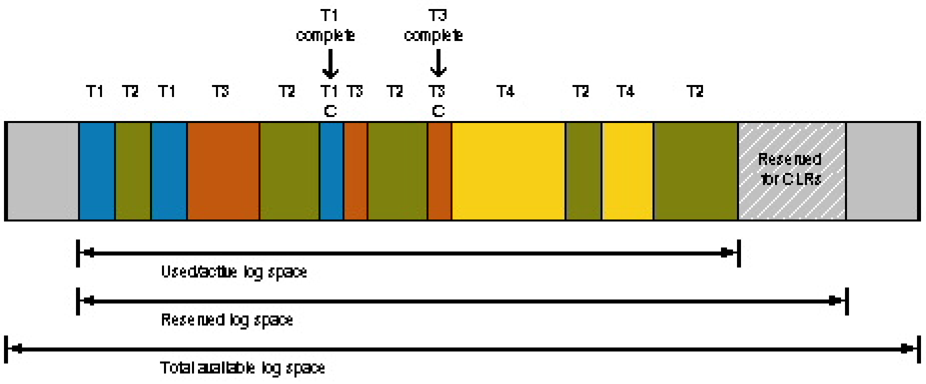
- Transactions are entered into the server but not committed. In this situation the log cannot be truncated because there is an active transaction.
- Replication Server is slow in reading the log which prevents truncating the log.
- A dump transaction has not been performed for a long period of time. Periodically dumping transactions can keep the size of the reserved space in the log limited and ensure that there is free space available in the log which allows the space freed after dump transaction to be reused for further logging activity.

You can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free log space. The **sp_thresholdaction** procedure can be used to free log space in the transaction log if the available free space falls below a preconfigured threshold. The recommended action is to define a trigger that will execute **dump transaction** once the log fall below the threshold. However, the **dump transaction** command cannot truncate the portion of the log beginning from of the oldest incomplete or active transaction in the log, since this portion is needed for recovery of the database. In this case, the oldest transactions can be aborted, depending on circumstances.

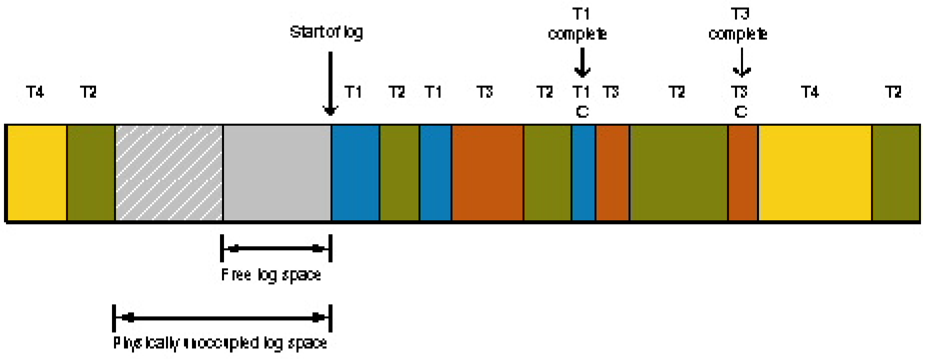
## Transaction Log Space

The transaction log contains sufficient information about each transaction to ensure that it can be recovered. This transaction information can be interleaved within the log.

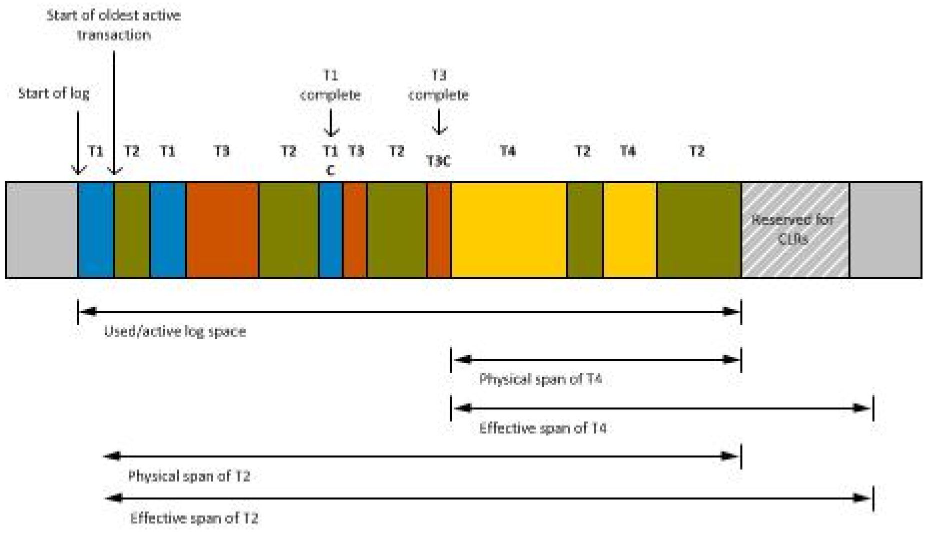
This transaction log shows the space allocation of the transaction log.



If a total allocated log segment is considered a contiguous logical space, then additional log records are appended to the end of the log. When there is no more space available at the end of the log, portions of the log that have been freed in the beginning of allocated space will be used to add records. This use of the freed portion of log can introduce a circular or wrapped around form within the allocated space.



The span of a completed transaction in the log is the portion of a log between begin log record of the transaction in the log and end log record of the transaction. The actual logging done by a transaction could be very minimal, however due to intermediate interleaved log records by other transaction, this span in terms of transaction log space could be considerable.



As transaction log space allocation is done in terms of pages, the span of an active transaction or incomplete transaction is also defined in terms of the number of log pages. There can be many active transactions at any point in the log. The portion of the transaction log occupied by span of an active transaction cannot be truncated with **dump transaction**. The total portion of the log which cannot be truncated with **dump transaction** is equal to span of the oldest active transaction

## loginfo

The **loginfo** function has been extended in Adaptive Server to further support managing transaction log space.

### Syntax

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

```
loginfo (dbid | dbname, option, option1]
```

### Parameters

- ***dbid* –** is the database ID.
- ***dbname* –** is the database name.
- ***option* –** is the specific information you need about the transaction log.

  These options have been added in Adaptive Server 15.7 SP110:

  - **oldest_active_transaction_pct** – returns a number from 0 to 100 indicating the span of the oldest active transaction in percentage of total log space.

- **oldest_active_transaction_spid** – returns the spid of the session having the oldest active transaction in the log of the Adaptive Server.
- **oldest_active_transaction_page** – returns the logical page number of start of oldest active transaction in the log. Returns 0 if there is no active transaction.
- **oldest_active_transaction_date** – returns the start time of oldest active transaction. Returns binary(8) number which needs to be converted to date as shown in the example below:

  ```
  select (convert(datetime, convert(binary(8),
      loginfo(4, 'oldest_active_transaction_date')), 109))
  ```
- **is_stp_blocking_dump** – returns 1 if there is a secondary truncation point before the start of the oldest active transaction, otherwise, returns 0.
- **stp_span_pct** – returns a number from 0 to 100 indicating the span of secondary truncation point to the end of log with respect to total log space.
- **can_free_using_dump_tran** – returns a number from 0 to 100 indicating the span of transaction log which can be truncated with the **dump transaction** command without having to abort oldest active transaction. If there is a secondary truncation point before the start of the oldest active transaction, then this is the span in the log (in percent) between the start of the log (first log page) and the secondary truncation point. If the secondary truncation point is not before the oldest active transaction, then this is the span in the log (in percent) between the start of the log (first log page) and start of the oldest active transaction.
- **is_dump_in_progress** – returns 1 if **dump transaction** command is in progress, returns 0 if no dump command is in progress.
- **database_has_active_transaction** – returns 0 if there are no active transactions in the log. Returns 1 if there is an active transaction in the log.
- **xactspanbyspid** – This option is to be used only with the third parameter, which is the SPID of the task. Returns the transaction span if the SPID has an active transaction in the log. Returns 0 otherwise.

These options are available in Adaptive Server 15.7 SP100 and later:

- **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.

**Note:** For a Mixed Log Data (MLD) database, this function returns a value equivalent to 0. The new options for this function are not supported or meant to be used for MLD databases.

## Examples

- **Example 1 –** Shows how to display transaction log information.

```
select loginfo(dbid, 'database_has_active_transaction'),
       loginfo(dbid, 'oldest_active_transaction_pct'),
       loginfo(dbid, 'oldest_active_transaction_spid'),
       loginfo(dbid, 'can_free_using_dump_tran'),
       loginfo(dbid, 'is_stp_blocking_dump'),
       loginfo(dbid, 'stp_span_pct')
```

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ----------- ------------------- ------------------ --------------- ------------ ----------------
     1           14              17                  7                 0             25            32
```

The function returns the transaction log information:

- 1 active transaction
- 14 is the SPID of the oldest transaction
- 17 percent of the log that is occupied by an active transaction
- 7 percent of the transaction log that can be freed by using the dump transaction command
- 0 blocking secondary truncation points
- 25 percent of the log that is occupied by the span of the secondary truncation point
- 32 percent of the log that is occupied

- **Example 2 –** Returns the amount of log space that is spanned for a particular transaction.

```
select loginfo(dbid, 'xactspanbyspid', spid)
spid      log_span_pct
----------------------
  15           2
```

### Permissions

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

## sp_xact_loginfo

**sp_xact_loginfo** provides the span of oldest active transaction in terms of percentage of total log space.

### Syntax

```
sp_xact_loginfo dbid [, vcharparam1] [, vcharparam2]
[, intparam1] [, intparam2]['span_pct] [, startpage]
[, xact_spid] [, starttime] [, firstlog_page] [, stp_page]
[, stp_pages] [, stp_blocking] [, canfree_without_abort_pct]
[, dump_in_progress] [, activexact] [, errorcode]
```

### Parameters

- **dbid –** is the database ID.
- **vcharparam1 –** varchar parameter indicating the mode. If **oldestactive**, the output parameter values are indicative of oldest active transaction. If **xactspanbyspid**, then output parameter values reflect values of active transaction for given spid.
- **vcharparam2 –** reserved for future use. Provide NULL as a value.
- **intparam1 –** integer parameter1 (SPID if *vcharparam1* = **xactspanbyspid**)
- **intparam2 –** integer parameter2
- **span_pct –** a value from 0 to 100. Indicates the span of transaction in percentage of total log space based on value of *vcharparam1*(output parameter).

- **startpage –** page number that is the start of the active transaction in the log based on value of vcharparam1. This page will hold the begin transaction log record of the active transaction.
- **xact_spid –** server process ID of the client having the active transaction based on *vcharparam1*.
- **starttime –** start time of active transaction based on *vcharparam1*.
- **firstlog_page –** server process ID of the client having active transaction based on *vcharparam1*.
- **stp_page –** secondary truncation point logical page number in the log. Returns -1 if replication is not active.
- **stp_pages –** returns the total number of log pages between the secondary truncation point and the oldest active transaction. Returns 0 if:
  - replication is not active
  - there is no active transaction in the log
  - there is no secondary truncation point before oldest active transaction
- **stp_blocking –** value of 0 or 1. 1 indicates that the secondary truncation checkpoint will block some portion for truncation beyond oldest active transaction span. Meaning that secondary truncation point is in between the start of the log and the start of oldest active transaction and replication agent must catch up. 0 indicates that aborting the oldest active transaction will free transaction log space without the secondary checkpoint blocking the abort.
- **canfree_without_abort_pct –** value from 0 to 100. Indicates the difference between **startlogpagenum** and **startxactpagenum** in terms of percentage of total log space. This portion can be truncated with the dump transaction command without aborting the oldest active transaction.
- **dump_in_progress –** returns1 if the dump transaction command is in progress, returns 0 if no dump command is in progress. Values of output parameters **firstlog_page** and **canfree_without_abort_pct** are not reliable. (output parameter).
- **activexact –** Boolean flag indicating that there are active transactions in the log.
- **errorcode –**
  - 0 there are no errors
  - 1 insufficient permission to execute
  - 2 error in opening dbtable. This could be due to various reasons including the dbid or database name given does not exist.
  - 3  cannot start xls session for log scan.
  - 4  there are no open transaction in the log against this database

**Note:** For a Mixed Log Data (MLD) database, this procedure returns values equivalent to 0 in output parameters. This procedure is not supported or meant to be used for MLD databases.

## Automating Transaction Log Management

Use cases to automate the management of the transaction log.

You can use **sp_thresholdaction** to identify the oldest active transaction.

You can use **sp_xact_loginfo** to monitor the longest running transaction per database or abort the oldest active transaction based on conditional criteria.

### Rescue Scenario Use Case

**sp_thresholdaction** can be used to identify the oldest active transaction and decide on action based on the information returned.

You can truncate the log to free up the space or abort the oldest active transaction or both based on the defined criteria. This use case assumes that the oldest active transaction span needs to be watched or limited only when free space in log segment falls beyond threshold.

```
create procedure sp_logthresholdaction
        @dbname         varchar(30),
        @segmentname    varchar(30),
        @space_left     int,
        @status         int
as
declare
        @oats_span_pct           int,
        @dbid                    int,
        @at_startpage            bigint,
        @firstlog_page           bigint,
        @canfree_withoutabort    int,
        @stp_blocking            int,
        @stp_pagebig             int,
        @dump_in_progress        int,
                @oat_spid               int,
                @oat_starttime          datetime,
                @activexact             int,
                @free_with_dumptran_pct int,
                @errorcode              int,
                @xactspan_limit         int,
                @space_left_pct         int,
                @dumptrancmd            varchar(256),
                @dumpdevice             varchar(128),
                @killcmd                varchar(128)

        select @dbid = db_id(@dbname)
        select @dumpdevice = “/ferrum_dev1/db1.tran.dmp1”

        select @free_with_dumptran_pct = 5
/*
** attempt dump tran only if it can free at
** least 5 percent of log space
*/
        select @xactspan_limit = 20
/*
```

```
** optionally also kill oldest active transaction even after
** dump tran if its exceeds 20 percent in the log
*/

    select @space_left_pct = logspace_pagestopct(@dbid, @space_left, 0)
    print “Space left in log segment ‟ + @space_left_pct + ‟ percent.”

    select @dump_in_progress = 1
    while (@dump_in_progress> 0)
        begin    -- {
            exec sp_xact_loginfo@dbid,
            “oldestactive”,
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output

if (@dump_in_progress> 0)
begin
    sleep 30
    continue
end
select @killcmd = “kill ‟ + @xact_spid + ‟ with status_only”

if (@canfree_withoutabort>@free_with_dumptran)
begin
     select @dumptrancmd = “dump tran ” + @dbname + ‟ on ‟ + @dumpdevice
     exec(@dumptrancmd)
/*
** Optionally, also abort oldest active transaction.
*/
        if ((@stp_blocking = 0) and
        (@oats_span_pct> @xactspan_limit))
        then
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
            exec(@killcmd)
        end
else
/*
** Some diagnostic information can be printed or warning actions
** can be executed here before aborting the transaction.
*/
    exec(@killcmd)
    end
    end -- }
```

### Monitoring Use Case

**sp_xact_loginfo** can be used for periodically monitoring the longest running transaction per database.

For example a stored procedure can be formed around **sp_xact_loginfo** in which it populates the tables with the oldest active transaction information and populates a user defined table. The execution of this stored procedure can be periodic, at desired frequency through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table
** for monitoring oldest active transactions with following
** table definition:
**
**    create table oldest_active_xact(
**           dbid                     int,
**           oldestactivexact_span    int,
**           startxactpagenum         int,
**           spid                     int,
**           xactstarttime            varchar(27),
**           startlogpagenum          int,
**           stppage                  bigint,
**           sec_truncpoint_block     int,
**           can_free_wo_kill         int,
**           dump_in_progress         int,
**           nolog                    int,
**           username                 varchar(30) null)
*/
create procedure sp_record_oldestactivexact
@dbname    varchar(30)
as
declare     @dbid int,
            @oats_span_pct int,
            @oat_startpage bigint,
            @firstlog_page bigint,
            @canfree_withoutabort int,
            @stp_blocking int,
            @stp_page bigint,
            @dump_in_progress int,
            @oat_spid int,
            @oat_starttime varchar(27),
            @activexact int,
            @free_with_dumptran_pct int,
            @errorcode int,
            @username varchar(30)

select @dbid = db_id(@dbname)

exec sp_xact_loginfo @dbid,
            'oldestactive',
            NULL,
            0,
            0,
            @span_pct = @oats_span_pct output,
            @startpage = @oat_startpage output,
            @xact_spid = @oat_spid output,
            @starttime = @oat_starttime output,
            @firstlog_page = @firstlog_page output,
            @stp_page = @stp_page output,
            @stp_blocking = @stp_blocking output,
            @canfree_without_abort_pct = @free_with_dumptran_pct output,
            @dump_in_progress = @dump_in_progress output,
            @activexact = @activexact output,
            @errorcode = @errorcode output

if (@activexact = 1)
    begin
        print "activexact is true"
        select @username = suser_name(sysproc.uid)
                from master..systransactions systran,
                master..sysprocesses sysproc
                      where systran.spid = @oat_spid
                and systran.spid = sysproc.spid
        insert into oldest_active_xact
            values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
                @oat_spid,
                @oat_starttime,
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,@username)
    end
else
    begin
        print "activexact is false"
        insert into oldest_active_xact values(
                @dbid,
                @oats_span_pct,
                @oat_startpage,
                @oat_spid,getdate(),
                @firstlog_page,
                @stp_page,
                @stp_blocking,
                @free_with_dumptran_pct,
                @dump_in_progress,
                @activexact,NULL)
end
```

## Monitoring and Control Use Case

In addition to monitoring, the action of aborting the oldest active transaction based on conditional criteria can also be implemented in **sp_xact_loginfo** which is run periodically through job scheduler.

```
/*
** Stored procedure assumes presence of a pre-created table for
monitoring
** oldest active transactions with following table definition:
**
**    create table oldest_active_xact(datetimetime_of_recording,
**            dbid                         int,
**            oldestactivexact_span        int,
**            spid                         int,
**            username                     varchar(30),
**            startxactpagenum             int,
**            startlogpagenum              int,
**            xactstarttime                datetime,
**            can_free_wo_kill             int,
**            sec_truncpoint_block         int,
**            nolog                        int,
**            action_taken                 varchar(30))
**            lock datarows
*/
create procedure sp_control_oldestactivexact
           @dbname    varchar(30)
as
declare    @oats_span_pct             int,
           @dbid                      int,
           @at_startpage              bigint,
           @firstlog_page             bigint,
           @canfree_withoutabort      int,
           @stp_blocking              int,
           @stp_pagebig               int,
           @dump_in_progress          int,
           @oat_spid                  int,
           @oat_starttime             datetime,
           @activexact                int,
           @free_with_dumptran_pct    int,
           @errorcode                 int,
           @username                  varchar(30),
           @action_taken              varchar(30),
           @xact_maxspan_pct          int,
           @killcmd                   varchar(128)

    select @dbid = db_id(@dbname)
    select @xact_maxspan_pct = 20
    select @action_taken = “none”

exec sp_xact_loginfo @dbid,
        “oldestactive”,
        NULL,
        0,
        0,
@span_pct = @oats_span_pct output,
@startpage = @oat_startpage output,
@xact_spid = @oat_spid output,
@starttime = @oat_starttime output,
@firstlog_page = @firstlog_page output,
@stp_page = @stp_page output,
@stp_blocking = @stp_blocking output,
@canfree_without_abort_pct = @free_with_dumptran_pct output,
@dump_in_progress = @dump_in_progress output,
@activexact = @activexact output,
@errorcode = @errorcode output

    select @killcmd = “kill “ + @oldesactive_spid + “ with status_only”

    if (@nolog == 0)
    then
select @username = suser_name(systran.suid)
from master..systransactionssystran where systran.spid
=@oldestactive_spid
if (@oats_span_pct> @xact_maxspan_pct)
begin
    exec(@killcmd)
    select @action_taken = “transaction abort”
end
        insert into oldest_active_xact values(getdate(), @dbid,
@oats_span_pct, @oat_spid, @username, @oat_page, @firstlog_page,
@free_with_dumptran_pct, @stp_blocking, @activexact, @action_taken)
    else
        /*
        ** Just to cover possibility of no active transactions which have
        ** generated any log.
        */
        insert into oldest_active_xact values(getdate(), @dbid,
                    0, 0, NULL, 0, 0, 0, 0, 1, @action_taken)
    end
```

## Analyzing and Managing Transaction Log Space

Use the **loginfo** function to view and free transaction log space.

The system administrator can use the **loginfo** function to evaluate how the space of a transaction log is used and determine the type of actions possible to free space.

This example uses **loginfo** to show the transaction log at a particular point in time:

```
======================================
select loginfo(dbid, 'database_has_active_transaction') as has_act_tran,
       loginfo(dbid, 'oldest_active_transaction_pct') as Act_log_portion_pct,
       loginfo(dbid, 'oldest_active_transaction_spid') as OA_tran_spid,
       loginfo(dbid, 'can_free_using_dump_tran') as dump_tran_free_pct,
       loginfo(dbid, 'is_stp_blocking_dump') as is_stp_blocking,
       loginfo(dbid, 'stp_span_pct') as stp_span_pct
```

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ---------------
      1           19               38                  7                 0              45           52
```

```
(return status = 0)
```

This shows:

- `has_act_tran` = 1, indicates that the database currently has one active transaction.
- `OA_tran_spid` = 19, indicates that the system process ID of the oldest active transaction in the database is 19.
- `Act_log_portion_pct` = 38, indicates that 38 percent of the log space is occupied by the oldest active transaction.
- `dump_tran_free_pct` = 7, indicates that 7 percent of the transaction log that can freed using **dump transaction**.
- `is_stp_blocking` = 0, indicates that there is no secondary truncation point preventing the use of **dump transaction** to free space.
- `stp_span_pct = 45`, indicates that there is a secondary truncation point spanning 45 percent of the transaction log.
- `log_occupied_pct` = 52, indicates that 52 percent of the total log space is currently occupied.

The available actions are:

1. The first step can be to use **dump transaction** to free the transaction log of the 7 percent shown by `dump_tran_free_pct = 7`. After freeing the space using **dump transaction**, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ----------------
      1          19              38                 0                 1              45           45

(return status = 0)
```

2. At this stage, `Act_log_portion_pct = 38`, indicates that 38 percent of the log space is occupied by the transaction with the system process ID of 19. You can wait for system process 19 to complete, or abort the transaction.

   If you decide to abort the transaction using the **kill** command (with or without status only option) as a measure to rescue the log, re-issuing the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ----------------
      0           0              0                 45                 0              0            45

(return status = 0)
```

3. The query shows that there are no active transaction in the system. You can free all 45 percent of the occupied log space using the **dump transaction** command. After **dump transaction** is executed, the output of the same query shows:

```
has_act_tran OAtran_spid Act_log_portion_pct dump_tran_free_pct is_stp_blocking stp_span_pct log_occupied_pct
------------ ------------ ------------------- ------------------ --------------- ------------ ----------------
      0           0              0                 0                  0              0            0

(return status = 0)
```

## Viewing the Span of a Transaction

The system administrator can view the span of a transaction started by a specific process.

In this example, the transaction is identified by system process ID 15 and the database ID is 4:

```
select loginfo(4, 'xactspanbyspid', 15)
    as xact_span_spid15_dbid4
```

```
xact_span_spid15_dbid4
 ----------------------
                     10
```

This indicates that system process 15 is an active transaction and the log transaction span is 10 percent.

## Viewing the Oldest Active Transactions

The system administrator can view the processes that are using the most log space.

This example shows the top three oldest active transactions having longest spans in the transaction log:

```
select top 3 convert(numeric(3,0),
      loginfo(db_id(), ‘xactspanbyspid’, t.spid)) as XACTSPAN,
          convert(char(4), t.spid) as SPID,
          convert(char(20), t.starttime) as STARTTIME,
          convert(char(4), p.suid) as SUID,
          convert(char(15), p.program_name) as PROGNAME,
          convert(char(15), p.cmd) as COMMAND,
          convert(char(16), p.hostname) as HOSTNAME,
          convert(char(16), p.hostprocess) as HOSTPROCESS
from master..systransactions t, master..sysprocesses p
where t.spid = p.spid
order by XACTSPAN desc
```

```
XACTSPAN SPID     STARTTIME      SUID PROGRAM COMMMAND  HOSTNAME
HOSTPROCESS
-------- ---- ----------------- ---- ------- -------- ----------
-----------
  38      19  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore4
26141
  20      20  Aug 5 2013 12:20AM  1    ISQL   WAITFOR linuxstore2
23467
  10      21  Aug 5 2013 12:21AM  1    ISQL   WAITFOR linuxstore6
4971
(return status =0)
```

## Parallel create index with Hash-Based Statistics for High Domains

Parallel **create index** in Adaptive Server 15.7 SP110 and later supports hash-based statistics gathering on high domain minor attribute columns of the index (that is, columns with 65536 or more unique column values).

Hash-based statistics gathering allows a database to immediately use newly created indexes without requiring the index scan from subsequent **update index statistics** *tab_name index_name* **with hashing** commands, which gather column statistics on the minor attributes of the index for query optimization (**update statistics** supports only serial hash-based statistics gathering).

The index must have more than one column for this feature to have effect.

**Note:** The major attribute of an index (that is, the first column of the index) continues to use legacy sort-based statistics gathering, whether or not you enable hash-based statistics gathering.

Parallel **create index** in earlier versions of Adaptive Server supported only hash-based statistics gathering on low-domain minor attribute columns of a composite index.

Each parallel thread creates its respective portion of the index, similar to earlier versions of **create index**. However, in versions 15.7 SP110 and later, while the index rows are being processed, hash based statistics gathering is invoked on each minor attribute of each index row. Each thread has a thread-local hash table for each column, so that the amount of `tempdb` buffer cache used increases proportionally by the number of parallel threads specified. In the case of high-domain, hash-based statistics gathering, additional memory is required to produce the final histogram .

The **max_resource_granularity** value limits the amount of memory used by all threads. If this limit is exceeded, one column is selected as a "victim" and the memory recycled is used to continue processing the remaining columns. If a victim is chosen due insufficient `tempdb` cache resources, the query processor does not generate statistics for the respective column. Typically, the high domain histograms created by parallelism is not the same as the histogram created by serial hash based processing, but in both cases the histogram cell weights are accurate for the respective cell boundaries.

## EMC PowerPath Device Fencing

Use the **SYBASE_MAX_MULTIPATHS** environment variable to override the number of paths iterated when PowerPath is enabled.

On the AIX platform, the method used for fencing EMC PowerPath devices is that the path of each device is fenced separately, which can result in performance issues during startup.

To increase startup performance when PowerPath is enabled, set the **SYBASE_MAX_MULTIPATHS** environment variable to 1, which will cause Adaptive Server to fence each device only once and allow PowerPath to broadcast the fencing operation to all of the paths.

- ksh: **export SYBASE_MAX_MULTIPATHS=1**
- csh: **setenv SYBASE_MAX_MULTIPATHS 1**

# SQL Standard for NULL Concatenation

Use **set sqlnull on** to implement SQL standard for NULL concatenation.

Standard SQL requires that string concatenation involving a NULL generates a NULL output. Adaptive Server evaluates a string concatenated with NULL to the value of the string. A string concatenation involving a NULL is treated as a string with a 0 length and an empty string ("") is interpreted as a single space.

Adaptive Server SP110 allows you to use the **set sqlnull** option to implement SQL standard for NULL concatenation.

This example, based on the table `staff_profile`, demonstrates the different output generated using the **sqlnull** option:

```
create table staff_profile(id int, firstname char(10) NULL,
surname char(10) NULL, city varchar(10) NULL, country varchar(10)
NULL )
go
insert staff_profile values(001, 'Tom', 'Griffin', 'Dublin', 'US')
insert staff_profile values(002, 'Kumar', NULL, 'Pune', 'India')
insert staff_profile values(003,  NULL , 'Kobe', 'Tokyo', NULL)
insert staff_profile values(004, 'Steve', 'Lewis', 'London', 'UK')
insert staff_profile values(005, 'Hana', 'SAP', NULL, 'Germany')
insert staff_profile values(006, 'Wei', 'Ming', 'Shanghai', 'China')
insert staff_profile values(007, 'city-state', '  ', 'Singapore',
'')
```

Output with the default value of **set sqlnull off**:

```
set sqlnull off
select id, rtrim(firstname) + '' + rtrim(surname) name, rtrim(city) +
'' + rtrim(country)
location from staff_profile

id           name           location
-------- ------------ ---------------------
1 Tom Griffin             Dublin US
2 Kumar                   Pune India
3 Kobe                    Tokyo
4 Steve Lewis             London UK
5 Hana SAP                Germany
6 Wei Ming                Shanghai China
7 city-state              Singapore
(6 rows affected)
```

Output with **set sqlnull on**:

```
set sqlnull on
select id, rtrim(firstname) + '' + rtrim(surname) name, rtrim(city) +
'' + rtrim(country)
location from staff_profile
```

```
id           name           location
-------- ------------ ---------------------
1 Tom Griffin            Dublin US
2 NULL                   Pune India
3 NULL                   NULL
4 Steve Lewis            London UK
5 Hana SAP               NULL
6 Wei Ming               Shanghai China
7 city-state             Singapore
(6 rows affected)
```

## Function Nesting in Expressions Increased to 32

The limit of 10 nested functions in an expression has been raised from 10 to 32.

# System Changes

Adaptive Server 15.7 SP110 adds changes to system procedures and functions.

## Commands

Changed Adaptive Server 15.7 SP110 commands.

### *set ins_by_bulk*

The **ins_by_bulk** parameter is improved for the **set** command, providing faster data loading by using parallel index updates for insert statements into a datarows-locked table with non-clustered indexes. The session-level syntax is:

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

### *load ... with headeronly*

Dumps that are created in a version of Adaptive Server using newer features may be incompatible for loading into earlier versions that do not support these newer features. The **load database** and **load transaction...with headeronly** commands report the database features that are contained in the dump, and the target version that allows the load of such dumps.

This is an example of the output from the **load transaction** command for a transaction log that contains activity from an optimized dump load with parallel index updates.

```
1> load tran pubs2 from '/linuxstore6_eng4/DBS/2k/UpgdDowngd/
pubs2.tran1.dmp' with headeronly
2> go

Backup Server session id is: 18. 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 'pubs21323503E6C  ' section number
1 mounted on disk file '/linuxstore6_eng4/DBS/2k/UpgdDowngd/
pubs2.tran1.dmp'

This is a log dump of database ID 4, name 'pubs2', from Aug 23 2013
4:26AM. ASE version: lite_744746-2/Adaptive Server Enterprise/15.7/EBF
XXXXX SMP SP110 /744746-2/x86_64/Enterprise Linux/. Backup
Server version: Backup Server/15.7 SP110/EBF XXXXX/P/Linux AMD Opteron/
Enterprise Linux/asecarina/4347/64-bit/OPT/T. Database page size is
16384.

Log begins on page 262147; checkpoint RID=Rid pageid = 0x42950; row num =
0x8a; previous BEGIN XACT RID=(Rid pageid = 0x40003; row num = 0x1);
sequence dates: (old=Aug 23 2013  4:24:28:740AM, new=Aug
23 2013  4:26:20:613AM); truncation page=272720; 9 pages deallocated;
requires database with 524288 pages.
```

```
Database log version=7; database upgrade version=35; database
durability=UNDEFINED.
segmap: 0x00000003 lstart=0 vstart=[vpgdevno=3 vpvpn=0] lsize=262144
unrsvd=259433
segmap: 0x00000004 lstart=262144 vstart=[vpgdevno=4 vpvpn=0] lsize=262144
unrsvd=260714

The database contains 524288 logical pages (8192 MB) and 524288 physical
pages (8192 MB).
dbdevinfo: devtype=0 vdevno=3 devname=pubs2_data physname=/
linuxstore6_eng3/DBS/asecarina/16k/pubs2_data.dat.16k devsize=2097152
dbdevsize=262144 vstart=0 devstat=2 devstat2=1
dbdevinfo: devtype=0 vdevno=4 devname=pubs2_log physname=/
linuxstore6_eng2/DBS/asecarina/16k/pubs2_log.dat.16k devsize=2097152
dbdevsize=262144 vstart=0 devstat=2 devstat2=1

This is the SYSDATABASES row of database ID 4, name 'pubs2' from the dump
header. Status:0x0205. Crdate:Aug 18 2013 11:32PM. Dumptrdate:Aug 23
2013  4:24AM. Status2:0x0000. Audflags:0x0000.
Deftabaud:0. Defvwaud:0. Defpraud:0. Def_remote_type:0. Status3:0x20000.
Status4:0x4000. Durablility:1. Lobcomp_lvl:0. Inrowlen:0. Dcompdefault:
1.
This is the SYSATTRIBUTES row for the  database 'pubs2' from the dump
header. Flmode:0x0000.
```

The output in **bold** lists the features contained in this dump, and the Adaptive Server version that can be used to load such a dump.

```
Features found in the database or transaction dump header:

ID= 4:15.7.0.000:Database has system catalog changes made in 15.7 GA
ID= 7:15.7.0.020:Database has system catalog changes made in 15.7 ESD#02
ID=11:15.7.0.100:Database has the Sysdams catalog
ID=13:15.7.0.100:Database has indexes sorted using RID value comparison
ID=14:15.7.0.110:Log has transactions generating parallel index
operations

----
```

The last row indicates that this dump contains activity from the optimized data load feature with parallel index updates. Such an activity can be safely recovered only in Adaptive Server Enterprise 15.7 SP110, or later. Loading this dump into an earlier version, such as Adaptive Server 15.7 SP100, is prohibited, and the **load** command fails.

Before creating a database or transaction dump, use **sp_downgrade** and **sp_downgrade_esd** and their associated recommendations to prepare your databases to be loadable in an earlier Adaptive Server version. Use the information from **with headeronly** to identify the features in a particular dump that may be incompatible with an earlier Adaptive Server version.

## System Procedures

New and changed Adaptive Server 15.7 SP110 system procedures.

### *sp_modifystats*

**sp_modifystats** adds these parameters:

```
 sp_modifystats [database].[owner].table_name, {"column_group" |
"all"},
. . .
modify_default_selectivity,
    {inequality | inbetween},
    {absolute | factor},
'value'
modify_unique
    {range | total },
    {absolute | factor},
'value'
```

Where:

- **modify_default_selectivity** – specifies the default selectivity value. Must be between zero and one, inclusive. **modify_default_selectivity** is one of:
  - **inequality** – indicates columns in which the predicate has an upper bound or a lower bound, but not both, and includes these range operators: > =, <=, >, <. The default value for **inequality** is .33
  - **inbetween** – indicates columns in which the predicate includes the upper bound and lower bound, and includes these range operators: > =, <=, >, <. The default value for **inbetween** is .25
  - **absolute** – ignore the current value and use the number specified by the value parameter.
  - **factor** – multiply the current statistical value by the value parameter.

  **Note:** You can use **modify_default_selectivity** only on an individual column, not a column group.

  Adaptive Server uses the default selectivity when an unknown constant prevents it from using a histogram to estimate selectivity of the respective predicate. The default selectivity for a search argument using inequality is 33%. **inequality** search arguments include columns for which there is an upper bound predicate or a lower bound predicate, but not both, and use the >=, <=, >, < range operators. The default selectivity for search arguments that include an **inbetween** search arguments is 25%. inbetween search arguments include columns that have both an upper bound predicate and a lower bound predicate, or use the **between** operator.
- **modify_unique** – allows you to modify the **range** unique or **total** unique values of a column or column group to the granularity specified in the *value* parameter.

- **range** – modifies the estimate for the number of unique values found in the range cells of the histogram. **range** does not include the frequency cells (that is, single-valued histogram cells). The estimate is represented as a fraction between 0.0 and 1.0, equal to:

  ```
  unique_range_values / (range_cell_rows * total rows_in_table)
  ```
- **total** – modifies the estimate of the number of unique values for the column or column group (including the NULL value). The optimizer uses this value to estimate **group by** and **distinct** cardinality. It is represented as a fraction between 0.0 and 1.0 where the 1.0/<unique count> is stored in the catalogs.
- **absolute** – ignore the current value and use the number specified by the value parameter.
- **factor** – multiply the current statistical value by the value parameter.

This example sets the default selectivity of inequality predicates with unknown constants (for example, `a1>@v1`) to 0.09:

```
sp_modifystats t10, a1, MODIFY_DEFAULT_SELECTIVITY, inequality,
absolute, "0.09"
```

This example sets the default selectivity for column a2 to use a value of 0.11 if you specify upper bound and a lower bound predicates with unknown constants (for example, `a2>@v1` and `a2<@v2`):

```
sp_modifystats t10, a2, MODIFY_DEFAULT_SELECTIVITY, inbetween,
absolute, "0.11"
```

This example modifies the range value for all columns for table t10 by a factor of 0.13:

```
sp_modifystats t10, "all", MODIFY_UNIQUE, range, factor, "0.13"
```

This example modifies the total unique value for all columns for table t10 to an absolute value of 0.14, which indicates there are (1.0 / 0.14) = 7.1428 unique values for each column in the table:

```
sp_modifystats t10, "all", MODIFY_UNIQUE, total, absolute, "0.14"
go
```

### *sp_xact_loginfo*

**sp_xact_loginfo** is a new procedure that provides the span of oldest active transaction in terms of percentage of total log space. See *sp_xact_loginfo* on page 14

## Functions

Changed Adaptive Server 15.7 SP110 functions.

### *shrinkdb_status*

**shrinkdb_status** adds these values for the **query** parameter:

- **current_object_id** – Object ID of the table being shrunk

- **current_page** – number of the page most recently, or currently, being moved
- **buffer_read_wait** – amount of time, in microseconds, spent waiting for buffers to be read
- **buffer_write_wait** – amount of time, in microseconds, spent waiting for buffers to be written
- **pages_read** – number of pages read by the shrink operation
- **pages_written** – number of pages written by the shrink operation
- **index_sort_count** – number of times the shrink operation sorted duplicated indexes

### *loginfo*

**loginfo** adds parameters to support managing transaction log space. See *loginfo* on page 11

### *db_attr*

**db_attr** has been expanded to include the **list_dump_fs** attribute.

You may not be able to load a database or transaction dumps that are generated in a later version of Adaptive Server into an earlier version. The features that are in use in a database, and objects that are created using newer features, are captured in database and transaction dumps. Before generating such dumps, use the **list_dump_fs** attribute to identify the features to be included in future dumps.

The following example shows the various features that are in use for the *pubs2* database, and the target Adaptive Server version, which can safely load such dumps. The last line in **bold** indicates that the optimized data load with parallel index updates was executed in this database, and is contained in the transaction log.

```
1> select db_attr('pubs2', 'list_dump_fs')
2> go

Features found active in the database that will be recorded in a
subsequent dump header:

ID= 3: 15.7.0.007: Database has compressed tables at version 1
ID= 4: 15.7.0.000: Database has system catalog changes made in 15.7 GA
ID= 7: 15.7.0.020: Database has system catalog changes made in 15.7 ESD#02
ID=11: 15.7.0.100: Database has the Sysdams catalog
ID=13: 15.7.0.100: Database has indexes sorted using RID value comparison
ID=14: 15.7.0.110: Log has transactions generating parallel index
operations
```

Future dumps of *pubs2* will be loadable only in the target server version indicated. To load the dumps of such a database in a target Adaptive Server version that is earlier than the version listed, downgrade the database to remove the footprint of newer features.

# Index