

PUBLIC
SAP Adaptive Server Enterprise 16.0 SP04
Document Version: 1.0 – 2020-12-10

In-Memory Database Users Guide

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 **SAP ASE In-Memory Database Options** . . . 5

2 **In-Memory Databases** . . . 6

2.1 Cache and Buffer Support . . . 7

2.2 Durability Levels . . . 8

Temporary Databases and In-Memory Temporary Databases . . . 9

Multidatabase Transactions and Database Types . . . 10

2.3 Template Databases . . . 11

Alter the Database to Use a New Template . . . 12

2.4 Minimally Logged Commands . . . 12

2.5 Limits for In-Memory and Relaxed-Durability Databases . . . 12

System Procedures Support . . . 13

2.6 Manage In-Memory and Relaxed-Durability Databases . . . 15

Specify Named Caches for In-Memory Databases . . . 15

Verify Changes to the Configuration File . . . 17

Create In-memory Devices . . . 18

Create In-Memory Databases . . . 18

Create Disk-Resident Databases with Relaxed Durability . . . 19

Administer In-memory Databases . . . 20

2.7 Minimally Logged DML . . . 22

Database-Level Logging . . . 23

Table-Level Logging . . . 24

Session-Level Logging . . . 25

Additional Minimal Logging Rules . . . 26

Transactional Semantics . . . 26

Logging Concurrent Transactions . . . 27

Minimal Logging with ddl in tran Set to True . . . 28

Effects of Referential Integrity Constraints . . . 28

Multistatement Transactions in Minimally Logged Mode . . . 29

Stored Procedures and Minimally Logged DML . . . 30

Including set dml_logging in a Trigger . . . 32

Using Deferred Updates . . . 33

Obtaining Diagnostic Information . . . 34

2.8 Performance and Tuning for In-Memory Databases . . . 34

Configure In-Memory Storage Cache . . . 34

sp_sysmon Output for In-Memory Databases . . . 35

Monitoring the Default Data Cache Performance . . . 37

Organizing Physical Data for In-Memory Devices. 37
Performance Optimization for Low-Durability Databases. 38
Minimally Logged DML. 41
Dumping and Loading In-Memory Databases. 43
Tuning for Spinlock Contention and Network Connections. 43

3 In-Memory Row Storage. 47
3.1 On-Disk Row Storage Log. 49
3.2 Types of Rows Hosted by the IMRS. 50
3.3 Interaction Between ILM and IMRS. 50
Automatic Partition Tuning. 51
Determining the Number of Rows a Single Query Can Move to the IMRS. 52
3.4 Memory Requirements for IMRS. 54
3.5 Using the Garbage Collector to Reclaim Memory. 55
Statement Registration and the Garbage Collector. 56
3.6 Determining the Size of the IMRS Components. 57
3.7 Configuring SAP ASE for In-Memory Row Storage. 58
3.8 Data Row Caching. 60
Row Caching Strategies. 61
3.9 Inserts, Selects, Updates, and Deletes in an IMRS-enabled Database. 62
Additional Information on Issuing DMLs in an IMRS-Enabled Database. 64
3.10 Creating Databases that Use a Row Storage Cache. 66
3.11 Altering Database to Use In-Memory Row Storage. 67
3.12 Creating and Altering Tables to Use Row Storage Cache. 68
Creating Tables the Use Row Storage Cache and Deferred Allocation. 69
Altering Tables to Use Row Storage. 70
Creating Indexes on Tables with In-Memory Rows. 71
3.13 Dropping Tables and Indexes from the In-Memory Row Storage. 71
3.14 Administering the IMRS. 72
Managing the Workload with sp_wlprofiler. 72
Determining the imrslog Threshold. 76
Interactions with Other Database Options. 79
Using BCP in an IMRS Database. 81
Using LOB Columns in the IMRS. 81
Using Triggers in the IMRS. 82
Committing Transactions in an IMRS. 83
Using Pack Operations to Move IMRS Rows to the Page Store. 84
Archive Management for ODRS. 86
Dumping and Loading into an IMRS-Enabled Database. 87
Shrinking of IMRS Databases. 88
Dropping IMRS Devices. 89
Using Compression with IMRS-Enabled Databases. 89

Interactions Between IMRS-Enabled Databases and Other Server Options. . . . 89
Using IMRS Garbage Collection to Free Memory. . . . 90
3.15 Troubleshooting the IMRS System. . . . 91
Rectifying Lack of Memory in the IMRS Cache. . . . 92
3.16 Performance and Tuning the IMRS. . . . 97
3.17 Restrictions for IMRS. . . . 99

4 Multiversion Concurrency Control. . . . 101
4.1 On-Disk Multiversion Concurrency Control. . . . 108
Estimating Required Resources for Enabling On-Disk MVCC. . . . 109
4.2 Scan and Query Behavior at Snapshot Transaction Isolation Levels. . . . 113
4.3 Using Snapshot Isolation Levels. . . . 113
4.4 Considerations when Using Snapshot Isolation Levels. . . . 114
4.5 Using Snapshot Isolations on Tables with LOB Columns. . . . 115
Off-Row and In-Row Data Movement. . . . 117
Reclaiming Pages Occupied by LOB Versions. . . . 117
4.6 Creating and Altering Tables and Databases to Use Snapshot Isolation. . . . 118
Creating and Altering Tables for Row Storage MVCC. . . . 118
Creating and Altering Databases and Tables for On-Disk MVCC. . . . 119
Altering Existing Databases to Use On-Disk MVCC. . . . 120
Creating Tables that Use On-Disk MVCC. . . . 121
Altering Existing Tables to Use On-Disk MVCC. . . . 122
4.7 Connection Support for MVCC. . . . 123
Using IsolationMode Property for ODBC Clients with DSN Connection. . . . 126
4.8 Troubleshooting the On-Disk MVCC. . . . 127

1 SAP ASE In-Memory Database Options

SAP ASE offers two fully featured in-memory database options.

- In-memory database (IMDB) – a page-oriented, zero-disk footprint in-memory database that uses the SAP ASE buffer cache to provide in-memory storage for the entire database.
- In-memory row storage (IMRS) – an in-memory, row-oriented storage cache that stores frequently accessed data from "hot" tables or partitions. Older, infrequently accessed data are stored in the page-based buffer cache. Distinct from an IMDB, only part of a disk-resident database's data is stored in-memory in an IMRS, while large amounts of the database are stored on disk, accessed through the buffer cache.
- IMRS cache supports these features:
  - Data row caching of hot data rows
  - Indexes with hash caching
  - In-memory multiversion concurrency control (MVCC) – snapshot isolation using in-memory versioning in the IMRS
  - On-Disk Multi-Version Concurrency Control (on-disk MVCC) – snapshot isolation using on-disk versioning in temporary database version store
    In the following sections, unless explicitly specified otherwise, the term "snapshot isolation" refers to either form of MVCC

The IMDB and IMRS features require these licenses:

- ASE_IMDB license to enable fully in-memory IMDB.
- ASE_MEMSCALE license to enable an IMRS in a disk-resident database and to configure the features that require an IMRS.

2 In-Memory Databases

In-memory databases run entirely in a named cache (that is, in the SAP Adaptive Server Enterprise memory space), without using disk storage for data or logs. Because an in-memory database does not require I/O, its performance can be much better than a traditional, disk-resident database.

In-memory databases are not designed for recovery: their transaction logs are written to the cache and not to disk, and any data changes are lost if the database fails. In-memory databases perform transactional logging for runtime rollback, and for other operations such as firing triggers, deferred mode updates, replication, and so on.

Disk-resident databases write to disk, and ensure that the transactional properties of atomicity, consistency, integrity, and durability (known as the ACID properties) are maintained. Durability refers to the persistence of transactions after they have committed. A traditional SAP ASE database, also known as disk-resident, operates at full durability by writing its transaction log to disk when a transaction commits. This, along with data pages being written periodically to disk, ensures that all committed transactions are durable.

In-memory databases do not write data or log to disk, and trade the guarantee of transaction durability for performance improvements. In the event of a database failure, in-memory databases cannot be recovered. If your applications require data recoverability following a server failure or a normal shutdown, consider using a traditional SAP ASE database.

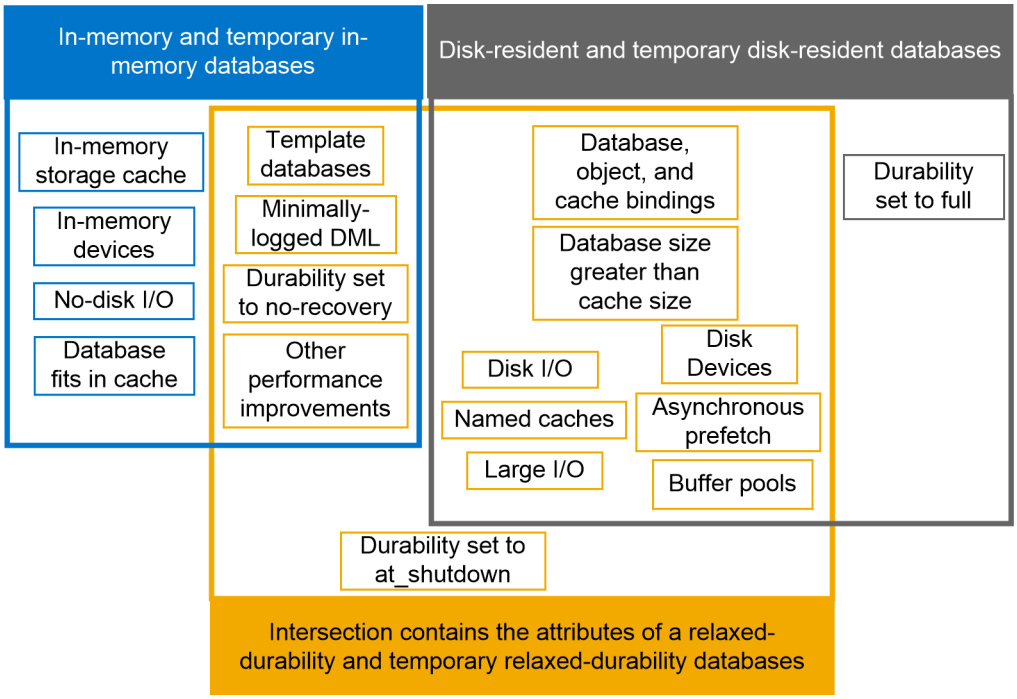
With support for relaxed durability, SAP ASE extends the performance benefits of an in-memory database to disk-resident databases. A disk-resident database operates at full durability to guarantee transactional recovery from a server failure. Relaxed-durability databases trade the full durability of committed transactions for enhanced runtime performance for transactional workloads.

The performance benefits of in-memory and relaxed-durability databases include:

- An in-memory database does not wait for I/O.
- Improved buffer and user log cache management, so you need not incur the overhead of user log cache flushes and buffer management when SAP ASE performs concurrent updates to the same data.
- Runtime strategies that may avoid flushing in the user-log cache to the transaction log when the transaction commits or aborts. These strategies reduce the contention on in-memory log pages.
- Support for minimally logged DML operations that use in-memory logging techniques improves the performance of large-volume DML operations.

You can create these types of databases (illustrated below):

- Disk-resident databases with durability set to full (this is the default, or traditional, SAP ASE database)
- User-created disk-resident temporary databases
- In-memory user databases with durability set to `no_recovery`
- User-created in-memory temporary databases with durability set to `no_recovery`
- Disk-resident relaxed-durability databases with durability set to `no_recovery` or `at_shutdown`



*Greater than or equal to the cache size

> **i Note**
>
> See your SAP Replication Server® documentation for information about using in-memory and relaxed-durability databases and DML logging in a replicated environment.

2.1 Cache and Buffer Support

Caches that host in-memory databases must be large enough to contain the entire database, and every page in the database must reside in the cache, without any buffer replacement or I/O to disk.

You cannot use caches that have been created:

- To host in-memory databases to bind other databases or other objects.
- For binding other databases or objects to host in-memory databases. Named caches used to host an in-memory database use a different structure, and are dedicated to in-memory databases.

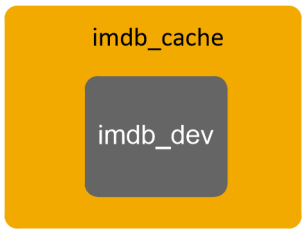
Use `sp_cacheconfig` to create the cache for the in-memory database. Use `disk init` to divide the cache into in-memory devices, which are similar to disk devices, and support segments. You can bind one or more logical segments to in-memory devices, allowing you to bind objects to individual segments.

Consider the following before you bind any objects to a cache for in-memory or relaxed durability databases:

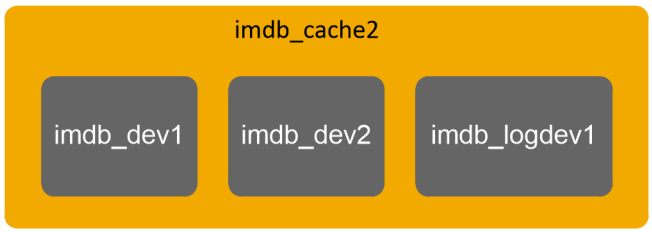
- Use named caches to bind entire relaxed-durability or full durability database. You may bind
  - Individual objects in an relaxed-durability database to a named cache, similar to binding individual objects in a regular database
  - Relaxed-durability database to a named cache (for example, the default data cache)

    - In-memory storage caches are similar to named caches, but are configured for efficient in-memory access. Individual objects in the relaxed-durability database to different caches
- Caching behavior for in-memory storage caches is similar to the caching behavior for regular caches.
- Use the same monitoring tools and tuning techniques you use to improve the performance of named caches to improve the performance of relaxed-durability databases bound to named caches.
- Because a single in-memory storage cache hosts the entire database, you do not bind a database or object to individual caches. Most existing cache manager monitoring and tuning apply to in-memory caches. See, "Monitoring Performance with sp_sysmon," in *Performance and Tuning Series: Monitoring with sp_sysmon*.

In-memory databases must be hosted by a single cache, but can reside on multiple in-memory devices created from that cache. This figure shows `imdb_cache` cache, which contains a single in-memory storage device, `imdb_dev`:



This figure shows `imdb_cache2` cache, which includes two in-memory data devices, `imdb_dev1` and `imdb_dev2`, and a log device, `imdb_logdev1`:



Use `create inmemory database` to create the in-memory database directly on the logical devices. If your installation does not use segments to limit space for objects, create the database on the in-memory device to be used for the entire in-memory storage cache. For finer granularity in space management for individual objects and threshold procedure support, create the database and log on separate in-memory devices.

Use `ddlgen` to generate database and object definitions for in-memory and relaxed-durability databases. See the *Utility Guide*.

Use `alter database` to change the layout of existing in-memory databases. See the *Reference Manual: Commands*.

2.2 Durability Levels

Data that you change in a committed, durable transaction survives after you restart the server following a system failure or `shutdown with nowait`.

The durability of a transaction in a traditional disk-resident database results from flushing the transaction log and the database pages to disk. In-memory databases provide no transactional durability after a server fails or impolite shutdown.

A relaxed-durability database offers two levels of durability from which you can select. The first level of durability is similar to an in-memory database: if the server fails, data is lost. The second level of durability is between that of a disk-resident database and that of an in-memory database: all transactions are completed and are persisted to disk only with a polite shutdown. This enables relaxed-durability databases to take advantage of many performance optimizations of in-memory databases.

Databases with a durability set to `no_recovery` or `at_shutdown`—whether they are in-memory or disk-resident—are referred to as low-durability databases. Data in low durability databases survives after a commit (provided you do not restart the server).

Use `create database with durability=<durability_level>` to set a database's durability level. SAP ASE supports `full`, `no_recovery`, and `at_shutdown` durability levels. See the *Reference Manual: Commands*.

This table shows the durability levels for databases describes the operations you can perform at each durability level.

Operation	`no_recovery`	`at_shutdown`	`full`
Create disk-resident databases	Yes	Yes	Yes
Create in-memory databases	Yes	No	No
Runtime rollback	Yes	Yes	Yes
Failure recovery	No	No	Yes
Database restored after a polite shutdown and restart.	No	Yes	Yes
Dump database to and load from archive.	Yes	Yes	Yes
Dump transaction log from device to archive. Load transaction log from archive to device.	No	No	Yes

Reduced durability and improved performance applies only to relaxed-durability databases using `at_shutdown` and `no_recovery`. You can bind relaxed-durability databases to named caches, for which the cache size can be smaller than the database size.

2.2.1 Temporary Databases and In-Memory Temporary Databases

Temporary databases implicitly use a durability of `no_recovery`.

You may create relaxed durability temporary databases with an explicit `no_recovery` durability to enhance performance, and you may alter existing temporary databases to explicitly set their durability to `no_recovery` after upgrade.

SAP ASE allows you to create and manage user-created `tempdb` groups in addition to managing the default `tempdb` group. User-created `tempdb` groups may include other user-created temporary databases, and support application and login binding.

You cannot remove the system `tempdb` from the `default` temporary database group. You cannot add system `tempdb` to any other user-created `tempdb` group.

Temporary database groups can mix disk-resident or in-memory temporary databases.

You can designate and administer user-created `tempdb` groups to contain only disk-resident or only in-memory temporary databases. Although SAP ASE does not explicitly impose any restrictions, by controlling the membership to the `tempdb` groups, you can assign specific disk-only or in-memory-only `tempdb` groups to specific logins or applications.

> **i** Note
>
> SAP ASE includes a 90-day trial period, during which you can try the in-memory database feature. However, if you use this feature, at the end of the trial period, you must:
>
> - Purchase an in-memory license, or
> - Purchase an SAP ASE license that does not include the in-memory feature. In this case, you must drop and recreate the user-defined temporary databases without using the explicit `durability` parameter.

2.2.2 Multidatabase Transactions and Database Types

A database's durability affects transactions that span multiple databases.

Low-durability databases—user-created temporary databases, in-memory databases, or disk-resident relaxed-durability databases—can participate in a multi-database transaction, even if the coordinating database uses full durability. Transaction are not allowed to span other full-durability databases.

Table 1: Database participants in a multidatabase transaction

Coordinating Database	Participant Database	Multidatabase Transaction
Full durability database	Temporary, in-memory, relaxed-durability, or temporary in-memory database	Allowed
Temporary, in-memory, relaxed-durability, or temporary in-memory database	Temporary, in-memory, relaxed-durability, or temporary in-memory database	Allowed
Temporary, in-memory, relaxed-durability, or temporary in-memory database	Full-durability database	Transaction are not allowed to span other full-durability databases

2.3 Template Databases

You can use a database other than `model` as the template for an in-memory database to load reference data (for example, tables and stored procedures) into the database created with the template.

The template database must be an existing disk-resident user database with `full` durability. Using templates to create a database and minimally logged DML are supported only on low durability databases.

You cannot use templates to create relaxed-durability databases that use a durability of `at_shutdown`. You can use templates only for databases that have a durability of `no_recovery`.

Use the `create database use <database_name> as template` command to specify a database other than `model` as a template for any database that has been created with the `no_recovery` parameter.

Once you create a database that uses a database other than `model` as its template, SAP ASE re-creates the dependent database, using data from the template database, when the server restarts.

When you restart the server, template databases are recovered before databases that use them as templates. If SAP ASE cannot recover a template database, databases that depend on that template database cannot be re-created.

SAP ASE applies the attributes of the template database when you create the dependent database. Attributes you specify as part of the `create database` command override the template database's attributes. Database options and attributes for databases you create using the template persist when the server restarts, and when the database is re-created from the template. Any changes you make to attributes of the template database are not used when SAP ASE re-creates the dependent database during subsequent restarts.

You cannot drop a database if any databases uses it as a template. You must first drop all databases that use this database as a template, or use `alter database` to detach the template database from all its dependent databases.

> **i Note**
>
> User-defined segments in template database may not function as expected after they are copied from the template database into an in-memory or relaxed-durability database. Segments direct space allocation onto specific database devices (or device fragments) by mapping the `syssegments` table to the `sysusages` table in the master database. Template databases may use a different mapping than the in-memory or relaxed-durability databases for which they are providing a template since the device layout for these databases may use a different number of devices and differently sized fragments. You must carefully plan and define the template database's user-defined segments before using these segments for in-memory or relaxed-durability databases.

Run `sp_helpdb` to report information about templates for:

- User databases – determine if, and for which databases, the user database has been used as a template. This section of `sp_helpdb` output shows that the `pubs2` database was used as a template for the `pubs3` and `pubs5` in-memory databases:

```
template_for
-------------------------------------------------
pubs3
pubs5
```

- Databases created from a template – determine which database was used as a template during creation (if it was not `model`). This section of `sp_helpdb` output shows that you used the `pubs2` database as the template for the `pubs3` database:

```
template
--------------------------------------------------
pubs2
```

2.3.1 Alter the Database to Use a New Template

SAP ASE does not change a database's existing data when you use `alter database` to change the template. SAP ASE uses data from the new template to re-create the database when you restart the server

You can change the template only for in-memory databases and for databases that use a durability of `no_recovery`; you cannot change the templates for system or traditional disk-resident databases using `full` durability.

2.4 Minimally Logged Commands

You can perform minimal logging for data manipulation language (DML commands) on a per-database, per-table, and session-specific basis so that minimal row and page changes, as well as page allocations and deallocations, are logged.

You can control DML logging at the database, table, and session levels by configuring commands such as `create database`, `alter database`, `create table`, `select into`, `set DML logging`, and `alter table`.

2.5 Limits for In-Memory and Relaxed-Durability Databases

Certain limitations apply to in-memory and relaxed-durability databases.

- The Cluster Edition does not currently support in-memory databases, relaxed-durability databases, template databases, or minimally logged DML.
- Replication Server does not currently support replication of in-memory databases, or databases with durability set to `no_recovery`.
- You cannot use an in-memory database as an archive database. SAP ASE recommends that you do not use an in-memory database as a scratch database.
- In-memory databases do not support queries using compatibility mode. Use compatibility mode in SAP ASE only on regular disk-resident database tables.
  If you have enabled compatibility mode and a query touches a table in an in-memory database, SAP ASE reverts to the native version 15.0 query optimizer and execution engine using the "restricted compatibility mode." Generally, this mode produces query plans similar to the plans in SAP ASE version 12.5. If you

notice a degradation in performance, SAP ASE recommends that you disable compatibility mode for this query.

- You cannot change the durability or logging level of in-memory or low-durability databases in the same command in which you are increasing the size of the database.
- `alter database`, when used to change either the `durability` or `minimal_logging` attribute of the database, automatically puts a database into single-user mode, and the command fails if it cannot acquire exclusive access to the database. To avoid this failure, manually put the database in single-user mode before running `alter database`.
- In-memory and relaxed-durability databases cannot participate in distributed transactions.
- In-memory and relaxed-durability databases cannot coordinate multidatabase transactions involving databases with full durability. When you execute system procedures that perform transactional updates when you run a stored procedure from an in-memory or relaxed-durability database, you see this error:

  ```
  Msg 3952, Level 16, State 2:
  Procedure 'sp_XX', Line 258:
  Command not allowed. You cannot start a multidatabase operation in database
  'master'
  after starting the master transaction in 'imdb1' as it may render the
  database 'master' unrecoverable.
  ```

  To run the system procedure:

  1. Execute it from a fully durable database. For example, to run `sp_XX` in `master`, enter:

     ```
     use master
     go
     exec sp_XX
     ```

  2. Use this format to reference the stored procedure from the current database (in-memory or relaxed-durability): `database_name.owner.<sp_name>`.
     For example, to run `sp_XX` in the `imdb_1` in-memory database, enter:

     ```
     use imdb_1
     go
     exec master.dbo.sp_XX
     ```

- From an in-memory database, you cannot use a proxy table or database to map to another in-memory database or disk-resident database object.
- From a disk-resident database, you cannot use a proxy table or database to map to an in-memory database or table.

2.5.1 System Procedures Support

Some stored procedures have been modified to support in-memory storage caches, in-memory devices, and in-memory databases.

System Procedure	Comments
`sp_addsegment`	Updated to manage space in in-memory databases.
`sp_addthreshold`	Updated to manage space in in-memory databases.

System Procedure	Comments
sp_bindcache	You cannot bind objects or databases to in-memory storage caches, and you cannot bind an in-memory database or objects in an in-memory database to any cache.
sp_cacheconfig	Creates, extends the size of, or drops, an in-memory storage cache.
sp_cachestrategy	The prefetch and MRU parameters do not apply to tables and indexes in in-memory databases.
sp_dbextend	Automatic database expansion is currently not supported for in-memory databases.
sp_deviceattr	The directio and dsync device attributes do not apply to in-memory devices.
sp_diskdefault	You cannot use sp_diskdefault to specify in-memory devices as a default device.
sp_downgrade	Supports downgrading an SAP ASE server containing in-memory or relaxed-durability databases, or databases using templates or minimal logging to an earlier version.
sp_dropdevice	Drops an in-memory device created from an in-memory storage cache.
sp_dropsegment	Updated to manage space in in-memory databases.
sp_dropthreshold	Updated to manage space in in-memory databases.
sp_extendsegment	Updated to manage space in in-memory databases.
sp_help	Reports properties, such as minimal logging attribute, for a table.
sp_helpcache	Displays properties of the in-memory storage cache, the in-memory database created on it, and details of free space on this cache.
sp_helpdb	Reports database properties such as durability, DML logging level, in-memory or not, use, if any, of a template database or as a template database.
sp_helpdevice	Reports the in-memory device properties created from an in-memory storage cache.
sp_modifythreshold	Updated to manage space in in-memory databases.

System Procedure	Comments
`sp_plan_dbccdb`	Sets up `dbccdb` for `checkstorage` execution in an in-memory database.
`sp_poolconfig`	Large I/O buffer pools are not supported in an in-memory database.
`sp_post_xpload`	Supports cross-platform operations for in-memory databases.
`sp_tempdb`	Supports login or application bindings for an in-memory temporary database.
`sp_unbindcache`, `sp_unbindcache_all`	You cannot unbind objects to the in-memory database itself from the host in-memory storage cache.

2.6 Manage In-Memory and Relaxed-Durability Databases

SAP ASE recommends that you use huge pages for in-memory storage cache.

Caches that hold in-memory databases must be large enough to contain the entire database. A cache that contains an in-memory database is called in-memory storage, and disables:

- I/O during runtime operations
- Buffer washing
- Buffer replacement and washing

Once created, divide the in-memory storage cache into one or more pieces, each of which holds a fragment of the database or log.

2.6.1 Specify Named Caches for In-Memory Databases

Use `sp_cacheconfig` with the `inmemory_storage` parameter to create the in-memory storage cache.

> **i Note**
>
> Before you create the in-memory storage cache, verify that the value for `max memory` is sufficient for the specified cache size. If `max memory` is insufficient, the server issues an error message.

For example, to create an in-memory storage cache named `imdb_cache`, enter:

```
sp_cacheconfig imdb_cache, '2G', inmemory_storage
```

> **i Note**
>
> For regular named cache, if the available memory size is less than the requested memory size for the cache, SAP ASE creates the cache with the reduced memory size. That means the cache is created successfully, but with a smaller size. However, if there is insufficient space to create the cache for an in-memory database, the command fails.

Each cachlet in an in-memory cache must contain a full allocation unit of 256 pages. If the amount of space you request does not completely full all the allocation units, the server reduces the amount of space allocated until all remaining cachlets are full. The page size of the server determines the amount of space reduced. For example, on a server using 2K logical pages, `sp_cacheconfig` allocates 96MB of 100MB requested:

```
 sp_cacheconfig tempdb_user_cache, "100M", "inmemory_storage",
"cache_partition=32"
go
sp_cacheconfig tempdb_user_cache
go
 Cache Name                          Status
         Type                                Config Value
         Run Value
 ---------------------------------- ----------------
         ---------------------------------- ----------------------------
         ------------------------
 tempdb_user_cache                   Active
         In-Memory Storage                      100.00 Mb
             96.00 Mb
(1 row affected)
                                               ------------ ------------
                            Total     100.00 Mb     96.00 Mb
===============================================================================
Cache: tempdb_user_cache,   Status: Active,   Type: In-Memory Storage
      Config Size: 100.00 Mb,   Run Size: 96.00 Mb
      Config Replacement: none,   Run Replacement: none
      Config Partition:           32,   Run Partition:           32
 IO Size          Wash Size                  Config Size
         Run Size                  APF Percent
 ---------------- ------------------------- ------------------------
         ------------------------ ------------------
     2 Kb                  0 Kb                  96.00 Mb
             96.00 Mb                  0
(return status = 0)
```

However, on a server that uses 8K logical pages, `sp_cacheconfig` allocates 64MB of the 100MB requested:

```
 sp_cacheconfig tempdb_user_cache, "100M", "inmemory_storage",
"cache_partition=32"
go
sp_cacheconfig tempdb_user_cache
go
 Cache Name                          Status
         Type                                Config Value
         Run Value
 ---------------------------------- ----------------
         ---------------------------------- ----------------------------
         ------------------------
 tempdb_user_cache                   Active
         In-Memory Storage                      100.00 Mb
             64.00 Mb
(1 row affected)
                                               ------------ ------------
                            Total     100.00 Mb     64.00 Mb
===============================================================================
Cache: tempdb_user_cache,   Status: Active,   Type: In-Memory Storage
```

```
      Config Size: 100.00 Mb,   Run Size: 64.00 Mb
      Config Replacement: none,   Run Replacement: none
      Config Partition:           32,   Run Partition:           32
 IO Size          Wash Size                  Config Size
         Run Size                 APF Percent
 ---------------- ------------------------- ------------------------
         ------------------------ ------------------
     8 Kb                  0 Kb                  64.00 Mb
            64.00 Mb                  0
(return status = 0)
```

Consequently, if you issue a command to create a 100MB cache: For this issue to show up, it requires larger page size and large cache partition values. in this example, for an 8K server, 2M of space is needed for an allocation unit of 256 pages. With 32 cache partitions, that means that the the inmemory cache has to be divisible by 64 (32 x 2). Hence the 100M is rounded down to 64M.

2.6.2 Verify Changes to the Configuration File

Verify that the configuration file (`$SYBASE/server_name.cfg`) correctly specifies in-memory storage cache information.

Each in-memory storage cache includes a heading in the configuration file that is labeled "Named Cache: `<cache_name>`."

The `Named Cache` entries include:

- `cache size` – size of the cache must be large enough to hold the entire in-memory database.
- `cache status` – set to "in-memory storage cache."
- `cache replacement policy` – set to "DEFAULT" or "none."
- `local cache partition number` – number of the local cache partitions or "DEFAULT."

An entry for a cache named `imdb_cache` looks similar to:

```
[Named Cache:imdb_cache]
cache size = 2G
cache status = in-memory storage cache
cache replacement policy = none
local cache partition number = 8
```

2.6.2.1 Change Static Configuration Parameters for In-Memory Databases

Changes are made to static configuration parameters when the server is restarted.

Because in-memory databases are re-created when you restart the server (and any changes to them are lost), perform either of the following to make sure the data in the in-memory database is not lost when you restart the server:

- Make all static configuration changes before the in-memory databases are created, or,
- Dump the in-memory database to an archive, make the static configuration change, restart the server, and load the in-memory database from the archive.

2.6.3 Create In-memory Devices

Use `disk init` to divide an in-memory storage cache into smaller pieces called in-memory devices, which are used to create in-memory databases. SAP ASE recommends that you use the same naming convention for in-memory devices that you use for disk-resident devices.

Bind user- or system-defined segments with in-memory device logical names to control the space usage for objects bound to these segments.

> **i** Note
>
> You cannot use regular named caches to create an in-memory device. That is, you must use the `type=inmemory` parameter for `disk init` to create in-memory devices.

The syntax to create an in-memory device is:

```
disk init name = <device_name>
    physname = {"<physical_name>" | "<cache_name>"}
. . .
[, type = "inmemory"]
```

where `<device_name>` is the logical name of the in-memory device, `<cache_name>` is the name of the in-memory storage cache created with `sp_cacheconfig`, and `inmemory` indicates the device is used for an in-memory database.

For example:

```
disk init name = imdb_cache_dev,
physname = "imdb_cache" ,
size = "50M",
type = "inmemory"
```

2.6.4 Create In-Memory Databases

Use `create inmemory database` to create an in-memory database, using `model` or another user database as its template.

You can also create temporary databases as in-memory databases that reside entirely in in-memory storage. However, you cannot specify a template database for an in-memory temporary database.

When you use a user-database template to create an in-memory database, all users, permissions, objects, and procedures are copied from the template database to the in-memory database.

When you create an in-memory temporary database:

- The guest user is added to the temporary database.
- `create table` privileges are granted to public.

You cannot create system databases (for example, `sybsecurity`) as in-memory databases, because these databases must be updated in the event of a server failure.

When you create in-memory databases, you can specify that `insert`, `update`, `delete` and some bulk-copy-in operations are minimally or fully logged on a per-database, per-object, or a session-specific basis.

This example creates an in-memory database on a 2GB in-memory device. `with override` allows you to create the data and log segments on the same in-memory device (the only durability level supported for in-memory databases is `no_recovery`: attempts to use another durability level result in an error):

```
create inmemory database imdb1
on imdb_data_dev1 = '1.0g'
log on imdb_data_dev1 = '0.5g'
with override, durability = no_recovery
```

Related Information

Minimally Logged DML [page 22]

2.6.5 Create Disk-Resident Databases with Relaxed Durability

Set the durability level for relaxed-durability databases to `no_recovery` or `at_shutdown`.

You can create a relaxed-durability database on existing disks. A relaxed-durability database uses disk storage, so you must use `disk init` to create the device on which it resides.

This example creates the `pubs6` database with a durability level of `at_shutdown`:

```
create database pubs6
on pubs6_dev
with override, durability=at_shutdown
```

Related Information

Durability Levels [page 8]

2.6.6 Administer In-memory Databases

Once you create in-memory databases, administer them by resizing and deleting the in-memory storage caches, increasing their size, and performing dumps and loads.

2.6.6.1 Resizing In-Memory Storage Caches

Use `sp_cacheconfig` to increase the size of an in-memory storage cache at runtime.

For example, to increase the size of `imdb_cache` to 3GB, enter:

```
sp_cacheconfig imdb_cache, '3G', inmemory_storage
```

You can reduce the size of an in-memory storage cache using the same procedure. You can reduce the size of the in-memory storage cache only by the amount of space on the cache that is currently unused by an in-memory database—in other words, the in-memory storage cache cannot be smaller than the in-memory database.

A reduced cache size is created when you restart the server.

2.6.6.2 Deleting In-Memory Storage Caches

You must drop all devices and in-memory databases before you delete the in-memory storage cache.

Delete the in-memory storage cache by setting its size to zero:

```
sp_cacheconfig imdb_cache, '0g'
```

2.6.6.3 Increasing the Size of In-Memory Databases

Use `alter database` to increase the size of an in-memory database.

The devices on which you increase the size must be part of the same cache that is currently hosting the devices on which the database resides (that is, you cannot create an additional in-memory storage cache and increase the size of an existing in-memory database on this storage cache).

You cannot run `disk resize` to increase the size of in-memory database devices (as you would for a standard database device).

To increase the size of an in-memory database:

1. Use `sp_cacheconfig` to enlarge the in-memory storage cache.
2. Use `disk init` to create a second in-memory device on the enlarged in-memory storage cache.
3. Run `alter database` to extend the in-memory database onto the new database device you created in step 2.

Use the same steps that you use to increase the size of a standard disk-based database to increase the size of a relaxed-durability database.

2.6.6.4 Dumping and Loading In-Memory Databases

Use `dump database` and `load database` to dump to or from an archive device for in-memory and relaxed-durability databases. Dumping and loading in-memory or relaxed-durability databases requires no special parameters for the `dump` or `load` commands.

`load database` into an in-memory database loads the data directly into the in-memory storage cache.

You can:

- Dump and load databases across durability levels. For example, you can dump a database with a durability level of `full` to an in-memory database, which always has a durability level of `no_recovery`
- Perform dumps and loads across platforms.

You cannot:

- Perform `dump transaction` from in-memory or relaxed-durability databases because `load transaction` cannot perform required tasks using a transaction log that may contain log records that are not ordered.
- Load a dump from a database with durability set to `no_recovery` or `at_shutdown` into a version of SAP ASE that does not support in-memory or relaxed-durability databases. However, you can load a database dump from an earlier version of SAP ASE in to an in-memory or relaxed-durability database.

> **i** Note
>
> You must use the version of Backup Server that ships with the version of SAP ASE you use to dump and load in-memory or relaxed-durability databases.

2.6.6.4.1 Configure Number of Backup Connections

Dumping and loading an in-memory database requires the Backup Server to connect to SAP ASE.

The `load` command uses the same number of connections as there are stripes. The `dump` command uses same number of connections as there are stripes, plus an additional connection. Use `number of backup connections` to configure the maximum number of user connections the Backup Server can use.

Connecting the Backup Server to SAP ASE requires that the Backup Server interfaces file has an entry for SAP ASE.

2.6.6.5 Dropping In-Memory Databases

Use `drop database` to remove an in-memory database.

This example drops the `pubs6` database:

```
drop database pubs6
```

Dropping an in-memory database removes the database from the system tables, and relinquishes the in-memory storage cache, although the buffers and data in these devices remain. Dropping an in-memory database does not affect the in-memory storage cache on which it was created, which can be used for other in-memory databases.

2.6.6.6 Dropping In-Memory Devices

Use `sp_dropdevice` to remove in-memory devices. `sp_dropdevice` removes an in-memory device only if it is currently unused by any database.

You must first drop the database, then drop the in-memory device. Dropping an in-memory device deletes its entry from `sysdevices` and returns the memory to the cache on which it was created, where you can use it for any other purpose, including creating new in-memory devices. This example drops the device named `pubs6_device`:

```
sp_dropdevice 'pubs6_device'
```

2.7 Minimally Logged DML

You can perform minimal logging for DMLs on a per-database, per-table, and session-specific basis. To optimize the log records that are flushed to the transaction log on disk, SAP ASE can perform minimal to no logging when executing some data manipulation language (DML) commands.

Minimal to no logging is performed when executing `insert`, `update`, `delete`, and `slow bcp` on all types of low-durability databases, such as in-memory databases and low-durability databases that use the `at_shutdown` and `no_recovery` options.

The hierarchy of control for DML logging settings are:

- Database-Level Logging – by default, logging for DML is enabled at the database level for all tables. DML logging settings affect only user and temporary tables.
- Table-Level Logging – overrides the level of logging set at the database level, depending on how the table was created or altered.
- Session-Level Logging – overrides the logging level set at the table and database levels.

2.7.1 Database-Level Logging

Database-level support for enabling and disabling logging is provided primarily for temporary databases, where entire applications that do not rely on logging can run more efficiently, without changing the code for applications or procedures that create temporary tables.

You cannot change the logging mode of system databases, including the `model` database. You can change the DML logging mode of any user temporary databases to use minimal logging. However, before doing so, SAP ASE recommends that you are mindful of how changing to minimal logging affects rollback semantics on all applications that use temporary tables.

Minimally logged DML is allowed only in a database that has a durability level set to `no_recovery` or `at_shutdown`. You must set the database's `select into` option to `on` for minimal logging to take effect.

You can alter the default logging mode for a database only from the `master` database. In addition, the database you are altering must:

- Be in single-user mode for user databases
- Have the `dbo-use only` set to `true` (on) for temporary databases, so that only the database owner can use the databases

> **i Note**
>
> Tables that you create in the system-level `tempdb` always use full logging. The server ignores commands that include `dml_logging` (for example, the `with dml_logging` parameter or the session-level `set dml_logging` command) when you create tables in the system-level `tempdb`.

If the database is not already in the required mode, the server tries to put the database in that mode. If the attempt is unsuccessful, the server raises an error, prompting the user to explicitly put the database in its correct mode.

The syntax to change the DML logging mode at the database level is:

```
create [temporary] database <database_name>
    [on {default | <database_device> [= <size>]
            [, {<database_device> [= <size>]...]
    [log on {<database_device> [= <size>]
        [, {<database_device> [= <size>]]...]
    [with {override | <default_location> = "<pathname>"
    | [[,]durability = { no_recovery | at_shutdown | full} ]
    | [[,]dml_logging = {full | minimal} ]
    }...
]
    [for {load | proxy_update}]
```

To change the database-level setting of DML logging in an existing database, use:

```
    alter database <dbname>
    set dml_logging = {full | minimal}
```

See the *Reference Manual: Commands*.

Related Information

Transactional Semantics [page 26]

2.7.2 Table-Level Logging

Setting the DML logging option at the table level overrides the database-level setting, depending on how the table was created or altered. The default mode is to use the same setting as the database.

If minimal logging is not enabled at the database level:

- Minimal DML logging is done for tables that have `dml_logging` explicitly set to `minimal` via `create table` or `alter table`.
- DML is fully logged for all other tables.

If minimal logging is enabled at the database level:

- Full DML logging is done for tables that have `dml_logging` explicitly set to `full`.
- Minimal DML logging is done for all other tables.

The `default` logging setting of a table allows it to inherit—when the DML is executed—the then-current database-level setting for logging. The database administrator may want to periodically turn logging off, then back on again, at the database level. Then, only those tables with specific needs for `full` or minimal DML logging need to be controlled through explicit table-level settings.

You can execute minimally logged DML commands on a table only if the database has the `select into` database option turned on; otherwise, all DML commands are fully logged.

For any DML statement on a table where the corresponding trigger is enabled (for example, if an `insert` trigger is enabled when you execute an `insert statement`) SAP ASE performs full logging. This avoids situations where a trigger implements business rules and security mechanisms that need the log records while executing the trigger. To perform minimal logging, specific triggers must be disabled by the table owner before executing the DML statement.

If a DML statement is executed on an view that can be updated, and it eventually resolves to a DML statement on a base table that is eligible for minimally logged DML operations, DML statements executed on the view result in minimally logged DML on the base table. To control the logging mode on base tables that are updated through views, use `alter table` or `set dml_logging` to set the logging mode on the underlying table.

You can use `alter table` to change only the logging mode of user tables, and not of views or other objects.

Commands

To create a table with full or minimal DML logging, use:

```
create table <tablename> (
    <<rest of the column list specifications>>
)
lock <lock_scheme>
with { max_rows_per_page = <num_rows>
```

```
    , exp_row_size = <num_bytes>
    , reservepagegap = <num_pages>
    , identity_gap = <value>
    , dml_logging = {full | minimal}
    }
on <segment_name>
```

To create a table using `select into` so that the target table has DML logging enabled or disabled, use

```
select <column list>
into <table_name>
    [ <<external table specifications>> ]
    on <segment_name>
    [ <partition_clause> ]
    lock <lock_scheme>
    with { max_rows_per_page = <num_rows>
        , exp_row_size = <num_bytes>
        , reservepagegap = <num_pages>
        , identity_gap = <value>
        , dml_logging = {full | minimal}
        }
[ from_clause ]
[ where_clause ]
...
```

> **i** Note
>
> In version 15.5, you cannot unconditionally turn on logging on a table when the table owner created the table explicitly turning off logging. This prevents large DMLs from generating huge amounts of logging on a table that was created to use minimal logging.

2.7.3 Session-Level Logging

Session-specific setting of the logging option overrides the table-level and database-level setting of the logging option.

To enable or disable logging for DML in the current session, even when database-level, or table-specific DML logging is full, use:

```
set dml_logging {minimal | default}
```

See the *Reference Manual: Commands*.

Setting the DML logging to `minimal` affects only the logging mode on objects owned by the current session user. If the session user has the `sa_role`, the logging mode of all user objects is `minimal`.

Once you have set session-specific DML logging to `minimal`, `set dml_logging default` returns the logging mode currently in effect for the affected tables to the table's default logging mode, based on the table- and database-level settings.

You can use the `set dml_logging` command to perform minimal logging for a table, but you cannot use it to perform fully logged DML if the database owner or table owner has already set up the table to run with minimal logging.

The session-specific setting for the logging mode eventually determines whether a particular object is logged, given the various rules described earlier and in the following sections, regarding choice of the logging mode for a specific table.

- Upon a successful `set` command invocation, all tables in the current session become candidates for consideration on choice of the logging mode, subject to permissions and privileges.
- You can use the `set` command to change the logging mode only for user tables, and not for other objects like system tables, views and so on.
- Minimal DML logging requires that the `select into` database option is turned on, which requires a database owner or `sa_role` privilege.

2.7.4 Additional Minimal Logging Rules

In addition to the basic rules related to databases, tables, and sessions, there are rules that also affect minimal logging.

- You can use minimally logged DMLs only in in-memory or relaxed-durability databases. You cannot use them in databases that have full durability.
- The logging mode for a table in a multistatement transaction remains unchanged throughout the entire transaction.
- All DML commands are fully logged on tables that participate in declarative or logical referential integrity constraints.
- You can export the `set dml_login` option from a login trigger to a client session. However, unlike most `set` options, you cannot export `set dml_logging` from stored procedures or issue `execute immediate` to their callers, even when you enable `set export_options`.
- All DML commands after a savepoint is set executes with `full` logging even though the table would have otherwise qualified for minimal logging.
- Full logging is performed if any active triggers exist on the table. For DML to run in minimal-logging mode, disable any triggers before executing the DML statement.
- An optimizer selecting deferred-mode update processing overrides the minimal DML logging setting, and the DML is executed with `full` logging.
- To support log-based replication, DML on replicated tables always performs `full` logging.

2.7.5 Transactional Semantics

When operating in minimally logged mode, the atomicity of transactions is not guaranteed after a runtime rollback.

Because logging is incomplete at runtime, you cannot completely roll back the changes made by a failed command. All changes are applied to the database in commit mode. For example, you cannot roll back a transaction that deleted a number of rows; the changes to the deleted rows are already committed. If a transaction deletes rows from a page that is then deallocated, the page remains deallocated.

However, not logging changes does not interfere with locking the rows or pages affected. Even if a DML command runs in minimally logged mode with no rollback available, the locks on the affected rows and pages

are acquired and held until the end of the transaction, and are released only upon transaction completion, be it `rollback` or `commit`.

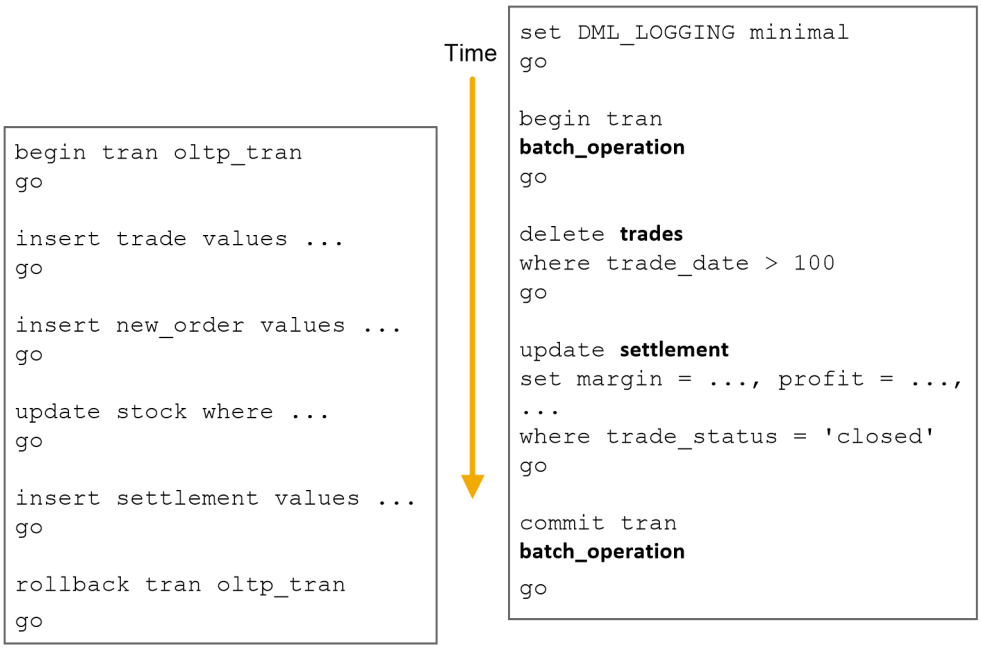
Issuing a `rollback` command when the transaction was executed with minimal logging on one or more tables raises a warning that SAP ASE is committing the transaction at the point where it encountered the rollback.

2.7.6 Logging Concurrent Transactions

The `set dml_logging` command affects the logging mode only in the current session, and only for DML statements on tables owned by the session's user.

This allows DML commands to execute concurrently from multiple transactions, where `full` logging is employed in one session executing concurrently with `minimal` logging from another session. Any rollback in the session executing with `full` logging is undone, whereas the changes are not undone in a rollback from the other session.

A common example of such a usage is the execution of a transactional system, running small transactions concurrently with large batch updates or deletes in minimally logged mode. Any errors in either session do not affect the transactional consistency of the other session. In the example below, the OLTP transaction runs with `full` logging, enabling complete recoverability, whereas the batch operation runs from another session with minimal logging.



2.7.7 Minimal Logging with ddl in tran Set to True

If the `ddl in tran` database option is set to `true`, you cannot execute a DML statement in minimally logged mode on a table that has already had any DDL operations executed against it in the same transaction. The DML commands in the transaction execute with `full` logging.

In the following transaction, the `sp_dboption` database option `ddl in tran` is set to `true`, and table `t1` is configured with minimal DML logging. However, at run-time SAP ASE executes the DML commands following the `drop index` command with full logging because it executed the `drop index` command with full logging:

```
sp_dboption pubs1, 'ddl in tran', true
go
begin tran
go
update t1 set ...
go
drop index t1.ind1
go
insert t1 values ...
go
insert t1 values ...
go
delete t1 where ...
go
update t1 where ...
go
rollback tran
go
```

The `create index` and `drop index` commands affect how minimal logging is selected when they are executed in the same transaction (that is, either of these commands would have caused SAP ASE to perform full instead of minimal logging in the script above).

2.7.8 Effects of Referential Integrity Constraints

When tables are involved in referential integrity constraints, rules apply for selecting minimal logging.

- All DML statements on a table with referential integrity constraints are always fully logged, overriding any logging settings arrived at for the table based on the database-level or session-specific settings.
- You cannot use `alter table` to change the logging mode of a table to `minimal` if the table includes referential integrity constraints.
- You cannot create any referential integrity constraints between tables that have minimal logging defined for them. For example, creating a foreign referential integrity constraint from a table to its primary table when minimal logging has been explicitly set on the primary table raises an error.

2.7.9 Multistatement Transactions in Minimally Logged Mode

There are rules that must be followed to use minimally logged mode DML in an explicit `begin transaction`, followed by multiple batches of DML statements.

- You may use DML statements that operate on different tables that are either in fully logged or minimally logged mode in a multi-statement transaction. You can use the `set dml_logging` command to change the logging mode partway through a transaction, but its effect on subsequent DML statements varies, depending on what other DML operations have already been executed in the same transaction. The following SQL transaction is allowed, assuming that the logging mode is `full` when you start the transaction:

  ```
  begin tran
  go
  delete t1 where ...
  go
  set dml_logging minimal
  go
  insert t1 values ...
  go
  update t2 where ...
  go
  commit tran
  go
  ```

  `delete t1` is performed with full logging, but `update t2` is performed in minimally logged mode. In the case of a rollback, the update is not undone, but the delete is rolled back.
- Once you execute a command with `full` or `minimal` logging on a table in a multi-statement transaction, subsequent commands on the same table must use the same logging (`full` or `minimal`), regardless of the session's `dml_logging` setting. In example above, the `insert t1` is executed with `full` logging because the earlier `delete` on `t1` which was executed with `full` logging.
- Conversely, resetting the session's logging mode to the default mode may not resume logging for a DML statement if the table was previously operated on in a minimally logged mode in the same transaction.
- Mixing the logging mode for different tables within the same transaction has different results for the tables involved if the transaction is rolled back. Changes to tables with full logging are rolled back, whereas changes to tables with minimal logging remain committed.
- The rules for logging mode choice and use of `set dml_logging` command inside stored procedures are identical to those for a multi-statement batch. If:
  - The procedure is run outside an explicit transaction, then each statement is executed as an individual transaction.
  - The procedure is run inside a transaction, then the same rules described above apply.
- There is no restriction on changing the logging mode inside a transaction and then executing `select` from a table that was previously operated on in the same transaction in a different logging mode. The `delete t1` is performed in logged mode, while the `update t2` is performed in minimally logged mode. Referencing the same table `t1`, for read, which was once operated on in fully logged mode when the logging mode is now `minimal` is not an error.

  ```
  begin tran
  go
  delete t1 where ...
  go
  set dml_logging minimal
  go
  update t2
  ```

```
where t2.c2 = (select c1 from t1 where ...)
go
commit tran
go
```

2.7.10 Stored Procedures and Minimally Logged DML

The DML logging setting in a session is inherited from called system procedures and the behavior setting for any `set dml_logging` command executed in a procedure is inherited by subprocedures.

After exiting from the procedure's scope, regardless of whether `set export_options` is `on`, the `dml_logging` setting in the parent session or parent procedure is restored.

These examples show how these rules are applied to affect the logging mode of tables inside a procedure.

Example: Example 1

In this example, the user executing the procedure is also its owner and the owner of all tables affected by this procedure:

```
create procedure p1 as
begin
    delete t1 where...
    set dml_logging minimal
    update t2 where...
end
go
set dml_logging default
go
exec p1
go
```

```
/*
** Exiting from the procedure restores the
** session's setting to what it was before
** calling the procedure, in this case, the
** logging mode will be back to DEFAULT
** (i.e. FULL).
*/
-- This will operate in logged mode now.
delete t2
go
```

1. When execution starts in the procedure `p1`, logging mode is `full`.
2. `delete t1` is performed in fully logged mode, and then the procedure's session-level logging mode is changed to `minimal`.
3. `update t2` is performed in minimally logged mode.
4. Upon exit from `p1`, when control returns to the outer SQL batch, logging mode setting is back to `full`. The next `delete t2` is performed in fully logged mode.

Example: Example 2

This example executes procedure `p1` with the session-level logging mode set to `minimal`. `delete t1` operates in minimally logged mode, and so does `update t2`. Once `p1` is done, the next `delete t2` also operates in minimally logged mode, as the logging mode has been restored to what it was before `p1` was called.

```
set dml_logging minimal
go
exec p1
go
-- This will operate in minimally logged mode.
delete t2
go
```

As the logging mode for DML statements inside a procedure is affected by the session-level setting of the calling session or procedure, SAP ASE recommends that you explicitly select the desired logging mode at the start of the procedure body. You may optionally set it back to `default` when the procedure finishes.

```
create procedure p1 as
begin
    set dml_logging minimal
    delete t1 where ...
    update t2 where ...
    -- Optionally, reset upon exit
    set dml_logging default
end
go
```

However, if the DML statements inside a procedure are executed mostly in a single-logging mode (for example, `full`) but must occasionally run in a different (minimally logged) mode, SAP ASE recommends that you control the logging DML statements with the session-level setting from the calling procedure or `isql` session instead of including the logging mode inside the body of the procedure.

Example: Example 3

The DML logging setting affects only those tables owned by the session's user. This affects procedures that must perform minimally logged DML on certain tables, but which are executed using the `exec proc` privilege by a user who does not own those tables.

In this example, Joe executes the procedure `mary.delete_proc`, which performs a `delete` on a table owned by Mary. Joe uses the `set` command to request minimal logging, but doing so affects the logging mode for tables owned by Mary in the procedure:

```
isql -Sservername -Umary -Pmaryspwd
create procedure mary.delete_proc as
begin
    delete mary.large_table where ...
end
go
grant exec on mary.delete_proc to joe
go
isql -Sservername -Ujoe -Pjoespwd
-- User 'joe' executes the following SQL:
--
set dml_logging MINIMAL
```

```
go
exec mary.delete_proc
go
```

Example: Example 4

SAP ASE does not permit the procedure owner, Mary, to allow minimal logging for certain statements when the procedure is executed by a user who does not own the table. The `set dml_logging` command inside a procedure applies only to those tables owned by the session's owner.

In this example, minimal logging does not apply to `delete mary.large_table`, but it dos apply to the `update joe.very_large_table` when user Joe executes the procedure with default logging settings.

```
isql –Sservername –Umary –Pmaryspwd
create procedure mary.delete_proc2 as
begin
    set dml_logging MINIMAL
    delete mary.large_table where ...
    update joe.very_large_table where ...
end
go
grant exec on mary.delete_proc2 to joe
go
isql –Sservername –Ujoe –Pjoespwd
exec mary.delete_proc2
go
```

If a procedure performs DML statements on tables owned by multiple owners, then, depending on which user executes the procedure, the set of tables against which minimally logged DML is executed changes. Only the table owner or user with `sa_role` can execute the procedure that performs the minimally logged DML on specific tables.

Recompiling a running procedure, or the cached plan of a previously executed procedure, does not affect the logging mode chosen at runtime for individual DML commands that appear in the procedure body. Any `set` commands that might change the logging mode for a table are taken into consideration at the start of execution of the DML statement.

2.7.11 Including set dml_logging in a Trigger

DML statements are fully logged if there is an active trigger for that DML operation on the table. If a trigger is created and logging is disabled on the table, a warning is raised, indicating that the DML statements will operate in fully logged mode.

However, the trigger is successfully created.

The logging mode of DML statements in triggers varies, depending on the user who fired the trigger. Tables owned by the same user who fired the trigger can be operated on in minimally logged mode. DML statements executed on tables that are not owned by the user who fired the trigger are executed in fully logged mode, unless these tables are explicitly set up for minimal logging.

Consider `delete_trig_m1`, a delete trigger on the object `m1` owned by user Mary. This trigger performs DML statements on other tables, such as `Joe.j2` and `Paul.p3`, in minimally logged mode. `delete` privilege has been granted on `Mary.m1` to user Sally.

```
Create trigger Mary.delete_trig_m1 FOR DELETE
On Mary.m1
as
Begin
    Delete Mary.min_logged_table_t3
    WHERE ...
    SET DML_LOGGING MINIMAL
    DELETE Paul.p3 WHERE ...
End
```

When user Sally executes the `delete` statement on `Mary.m1`, the trigger, `Mary.delete_trig_m1` is fired and privilege checks are performed on the appropriate tables on behalf of Sally. Because Mary owns the trigger, SAP ASE performs no permission checks for the `delete Mary.min_logged_table_t3` statement, where the DML statement is run in minimally logged mode. (The table has been defined for minimal logging through some other means.) Because Sally does not have the privilege to turn off `dml_logging` for `Paul.p3`, the `set` command does not take effect silently. The next statement, `delete Paul.p3 where...`, runs in fully logged mode. When user Mary executes the trigger, the DML logging behavior is the same (fully logged).

However, if user Paul executes the outer `delete` on `Mary.m1`, causing the trigger to fire, then both the statements in the body of the trigger execute in minimally logged mode.

Once the trigger fires, the outer DML statement must have been fully logged; attempts to perform minimally logged DML on the same outer table in the body of the trigger are ignored, and these DML statements are fully logged.

Logging mode cannot be disabled on views, so `instead of` triggers, which are currently only supported on views, are unaffected by the logging mode of the base tables referenced by the view, or by the session-level setting of the logging mode when the DML statements on the view with the `instead of` trigger executes. However, the rules for choosing logged or minimally logged modes on multiple tables, using of the `set` command inside the trigger body, and transactional semantics all apply for the DML statements inside the `instead of` trigger.

2.7.12 Using Deferred Updates

If a query optimizer picks a deferred-mode update for a table that qualifies for minimally logged operations, the minimally logged settings are overridden for that statement and the statement works in deferred mode, but with `full` logging.

Applications that generate DML statements in deferred mode with large transaction logging do not benefit from minimal logging. Once you execute a deferred-mode operation for multistatement transactions on a table, all subsequent DML statements on this table are fully logged.

2.7.13 Obtaining Diagnostic Information

Many rules interact to determine the logging mode of a table. Application developers need to know whether SAP ASE is currently generating minimally logged DML commands for a particular table, and the type of schema, constraints, session settings, and so on.

`object_attr` reports the table's current logging mode, depending on the session- table- and database-level settings. See the *Reference Manual: Building Blocks*.

Use `set print_minlogged_mode_override` to determine whether SAP ASE has overwritten your logging-mode choices (see the *Reference Manual: Commands*). `print_minlogged_mode_override` generates trace information to the session output, reporting on the statement for which the minimally logged mode of a table has been overwritten by other rules, such as presence of referential integrity constraints, deferred mode choice, name of the table affected, a description of the affecting rules, and so on. Enable this switch server-wide to capture diagnostic output from the entire application. Redirect this output to the error log using the `print_output_to_log` switch.

2.8 Performance and Tuning for In-Memory Databases

SAP ASE supports a wide variety of database and cache configurations for all database types using named caches and database bindings.

Consider the following when sizing or binding a cache:

- Bind large, disk-resident databases to smaller named caches. This is a standard SAP ASE configuration, with only a small portion of the database's contents in cache. Depending on the workload, much of the database page space may get recycled due to normal cache replacement policies.
- Make the cache size similar to the database size, which may reduce the recycling of cached pages. However, depending on the workload, writes to disk may be high.
- Bind entire temporary databases to a named cache that is large enough to host the entire temporary database. Default optimizations the server applies to temporary databases (and using strategies such as `delayed commit`) allow you to significantly improve the performance of fully cached temporary databases.

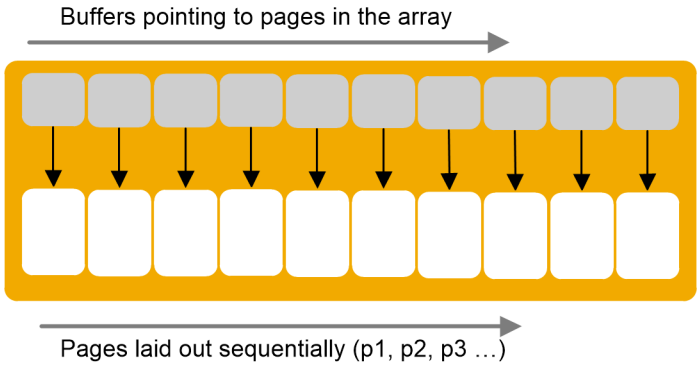
2.8.1 Configure In-Memory Storage Cache

Relaxed-durability databases, and objects in them, that are bound to any named cache use the same layout as regular caches.

The access pattern of the object bound to the cache dictates the layout for a regular cache. SAP ASE selects the least recently used (LRU) buffer and page pair for replacement with strict LRU cache replacement policy. SAP ASE loads pages for objects bound to the named cache that reside on disk (identified by the database, table, and index ID) into the cache when they are requested. These objects remain in the cache until SAP ASE selects them for replacement. SAP ASE determines the location in the cache where this page is read, according to the availability of free buffers or according to which buffers are replaceable, depending on the LRU or most-

recently used (MRU) strategy. The buffer that is used to hold a page in memory might vary, and the location of the buffer in the buffer cache may also vary in this cache configuration

Databases that use in-memory storage caches are bound to the cache, and all the objects and their indexes in this database use the same cache. The entire database is hosted in the cache. All the pages in the database, both allocated and unallocated, are hashed, so page searches in the cache should always find the required page. Pages are laid out sequentially, and because all the pages reside in the cache, the position of the buffer and page pair does not change.



In-memory storage cache supports only the default pool (which uses the server page size for any logical read or write on that page). In-memory storage caches do not need large buffer pools, which are used to perform large I/O from disk. Asynchronous prefetch, as a strategy to improve runtime I/O performance and to reduce cache misses while searching for a page, is not supported for in-memory storage caches.

Because they do not store any data in disk, in-memory storage caches:

- Do not use any replacement policy (the replacement policy is defined as `none` for each cache). The pages in the cache never change positions.
- Do not support I/Os, so the wash region and large pools are not supported.
- Need not perform buffer replacement. Occasionally, for private buffers, the default data cache is used.
- Need not dynamically hash buffers. All pages are loaded and hashed in the hash table when you create the database. In memory databases may need to hash buffers at runtime if you increase the size of the database with `alter database`.
- Need not perform asynchronous prefetch, because all pages reside in the cache memory.

Cache partitions are supported for in-memory storage caches, and reduce the contention on a single spinlock by distributing subsets of pages in an in-memory database to a partition, where each subset is controlled by a separate spinlock.

2.8.2 sp_sysmon Output for In-Memory Databases

Run `sp_sysmon` for in-memory databases to monitor their performance.

Cache Hits should always have a value of 100% for an in-memory database cache, and Cache Misses should always have a value of 0:

```
Cache: imdb
                              per sec      per xact       count   % of total
------------------------   ------------  ------------  ----------   ----------
Spinlock Contention                 n/a           n/a         n/a      50.9 %
```

```
Utilization                          n/a           n/a         n/a      99.9 %
Cache Searches
Cache Hits                       16954.4        5735.8     1152897     100.0 %
Found in Wash                        0.0           0.0           0       0.0 %
Cache Misses                         0.0           0.0           0       0.0 %
-----------------------   ------------  ------------  ----------   ----------
Total Cache Searches             16954.4         5735.8     1152897
```

In-memory caches should not have a wash region, as reported below:

```
Buffer Wash Behavior
   Statistics Not Available - No Buffers Entered Wash Section Yet
Cache Strategy
Cached (LRU) Buffers          21225.3         7180.7     1443321      100.0 %
Discarded (MRU) Buffers           0.0            0.0           0       0.0 %
```

See *Performance and Monitoring Series: Monitoring with sp_sysmon*.

The following `sp_sysmon` sections do not apply to in-memory storage caches because the entire database is bound to the cache and all database pages reside in the cache:

- Utilization
- Cache Hits
- Found in Wash
- Cache Misses
- Large I/O Usage
- Pool Turnover

These sections do not apply to in-memory databases because they do not perform buffer grabs:

- LRU Buffer Grab
- Grabbed Dirty
- Total Cache Turnover

These sections do not apply to in-memory databases because they do not define a wash region. Disk reads and writes do not take place in a steady state:

- Buffer Wash Behavior
- Cache Strategy

These sections do not apply to in-memory databases because they do not use large pools:

- Large I/Os Performed
- Large I/Os Denied
- Total Large I/O Requests
- Large I/O Detail

2.8.3 Monitoring the Default Data Cache Performance

Because all pages are hashed in an in-memory storage cache, tasks requiring a small number of private buffers for storing intermediate pages, for example during sorting, use the default data cache.

Use the `buf_imdb_privatebuffer_grab` monitor counter to determine if SAP ASE is temporarily using the default data cache for sort buffers:

```
buf_imdb_privatebuffer_grab  buffer_0           244              510525
```

This script collects monitor counters from all monitoring groups. It first clears all the monitor counters, samples the monitor counters for one minute, and reports values for all monitor counters updates for that monitoring interval.

To determine the current value for `buf_imdb_privatebuffer_grab`, run:

```
dbcc monitor ('clear','all','on')
go
dbcc monitor ('sample','all','on')
go
waitfor delay "00:01:00"
go
dbcc monitor ('sample','all','off')
go
dbcc monitor ('select','all','on')
go
select * from master.dbo.sysmonitors
where field_name = 'buf_imdb_privatebuffer_grab'
go
```

The value for `buf_imdb_privatebuffer_grab` indicates the default data cache's buffer usage from queries that require temporary buffers for intermediate sorting that are run against tables in an in-memory database. Evaluate the value for `buf_imdb_privatebuffer_grab` according to the size of the default data cache:

- If the number of buffers grabbed is very low compared to the size of the default data cache, queries run against tables in in-memory databases are not heavily using temporary buffers.
- If the number of buffers grabbed is a significant portion of the number of buffers in the default data cache, it indicates a heavy load on the default data cache for buffers used for queries running against in-memory databases. This generally occurs only when the default data cache is very small, for example, if it is using the default size of 8MB.
  Using a small default data cache may affect the performance of other applications that rely on the default data cache. Increase the size of the default data cache to accommodate both requests for temporary buffer usage that come from in-memory databases, and to accommodate other concurrent applications using the same cache.

See, "Monitoring Performance with `sp_sysmon`," in *Performance and Tuning Series: Monitoring with sp_sysmon*.

2.8.4 Organizing Physical Data for In-Memory Devices

Because in-memory databases do not use I/O, you need not consider device I/O characteristics (for example, the speed of the I/O device) when organizing physical data placement. You need not consider issues like

locating frequently accessed index pages on fast devices, or pages for other infrequently utilized objects (for example, text and image pages) on slower devices.

However, because in-memory databases remove other bottlenecks caused by disk I/O, other bottlenecks such as spinlock and latch contention may be high.

Consider the following when you organize physical data for in-memory devices:

- To reduce latch contention, use separate data and log devices (that is, do not configure in-memory databases for mixed log and data).
- To control log space consumption, use separate log devices.
- To restrict object space usage, organize devices in segments.
- To reduce contention for page allocation, use segments bound to different devices.

2.8.5 Performance Optimization for Low-Durability Databases

Low-durability databases have their durability set to `no_recovery` and `at_shutdown`.

Because low-durability databases are not recovered, they provide these performance benefits:

- Low-durability databases do not flush the user log cache (ULC) due to unpinning, therefore increasing transaction throughput in high-transaction systems.
  ULC unpinning occurs when two transactions attempt to update the same datarows-locked data page, causing the second transaction to flush the log records from the first transaction's ULC to `syslogs`, before the second transaction uses the data page. Increasing ULC flushes to `syslogs` increases log contention, adversely affecting transaction throughput in high-transaction systems.
  The Transaction Management section in `sp_sysmon` output shows that the values for ULC Flushes to Xact Log, by Unpin, and by Single Log Record (all actions associated with unpinning) are noticeably lower for low-durability databases than in full-durability databases. This increases the transaction throughput in high-transaction systems. This `sp_sysmon` output from a low-durability database shows no ULCs flushed from unpinning:

```
ULC Flushes to Xact Log    per sec       per xact    count  % of total
-----------------------  ----------   ------------  -------  ---------
Any Logging Mode DMLs
by End Transaction             40.3            0.0     2416   100.0 %
by Change of Database           0.0            0.0        1     0.0 %
by Unpin                        0.0            0.0        0     0.0 %
by Other                        0.0            0.0        0     0.0 %
```

- Low-durability databases do not perform transaction log flushing – during a `commit`, because transactions are not written to disk.
  Transaction log flushing is typically unnecessary for relaxed-durability databases; they do not require reliably recorded committed transactions because they are not recovered during a restart. SAP ASE typically flushes the transaction log when a transaction completes and, for relaxed-durability databases, during the buffer wash.
  The Device Activity Detail section in `sp_sysmon` output shows the number of writes for log devices of relaxed-durability databases. This shows the `sp_sysmon` output for a disk-resident, version 15.0.3 database:

```
Device:
```

```
/disk_resident/device/1503esd2/drdb_device.dat
drdb_device               per sec      per xact       count  % of total
-------------------   -----------  ------------  ---------  ----------
Reads
  APF                         0.0           0.0          0      0.0 %
  Non-APF                    14.3           0.0       2380      0.2 %
Writes                     9247.3           1.5    1544304     99.8 %
-------------------   -----------  ------------  ---------   ----------
Total I/Os                 9261.6           1.5    1546684     100.0 %
```

This shows the same output for a relaxed durability database. Because this is a relaxed-durability database, the writes are about 23 percent of the writes for the disk-resident database:

```
Device:
/relaxed_durability/IMDB/devices/ariesSMP/rddb_device.dat
rddb_device               per sec      per xact       count   % of total
-------------------   -----------  ------------  ---------   ----------
Reads
  APF                         0.0           0.0          0      0.0 %
  Non-APF                    14.7           0.0       1031      0.7 %
Writes                     2106.4           0.1     147446     99.3 %
-------------------   -----------  ------------  ---------   ----------
Total I/Os                 2121.1           0.1     148477     99.9 %
```

- Low-durability databases do not log transactions contained in the ULC. SAP ASE discards log records for transactions in low-durability databases when it commits transactions, if all log records for each transaction are fully contained within the ULC, and the transaction does not require the log records for any post-commit processing.
  Because you need not transfer the log records, the amount of contention on the log decreases, which increases throughput in high-transaction systems. Generally, logging transactions in the ULC favors smaller transactions because large transactions cannot contain all their log records in the ULC at commit time; large transactions that log more records than fit in the ULC must flush the ULC to `syslogs` when it fills.
  ULC flushing may also occur when small transactions are contained within the ULC at commit time because SAP ASE requires these log records to perform post-commit work. For example, ULC flushing occurs in any transaction that deallocates any space within the database when the transaction requires post-commit work.
  See “ULC Flushes to Xact Log” in, “Monitoring Performance with sp_sysmon,” in the *Performance and Tuning Series: Monitoring with sp_sysmon*.
- Low-durability databases do not flush partially logged changes to disk. SAP ASE must occasionally flush data pages for full-durability databases to disk because the corresponding log records do not fully describe the changes that took place. Some examples are:
  - Any index- or data page split for a table with a clustered index
  - A sort
  - A `writetext` command
  - Fast `bcp`
  - `alter table` to change a table’s locking scheme
  - `alter table` to change a table’s partitioning scheme or a table’s schema
  - A full table `reorg rebuild`
  - `alter table...unpartition`

  In-memory databases need not flush data pages to disk because they do not use disks. Relaxed-durability databases do not flush data pages to disk because there is no need for database recovery. Not having to flush data pages to disk can significantly improve performance when the number of disk I/Os would be high, for example, where changed pages are flushed to disk at the end of a `sort` operation.

Additional improvements to low-durability databases that are not directly related to their durability, occur internally, and require no administration:

- Improvements for updating the database timestamp (a frequent operation in a high-transaction database)
- Improvements to deletes using an index scan for data-only-locked tables
- Improvements for bulk inserts into a data-only-locked table with non-unique indexes

These improvements also apply to temporary databases that have durability explicitly set to `no_recovery` (through `create database` or `alter database`); they do not apply to temporary databases with durability implicitly set to `no_recovery`.

2.8.5.1 Tuning Checkpoint Intervals

In addition to determining the length of the recovery time, the `recovery interval in minutes` configuration parameter determines how frequently SAP ASE checkpoints a database.

In-memory databases are never checkpointed, but SAP ASE does checkpoint relaxed durability databases, flushing all modified buffers from disk according to `recovery interval in minutes`.

Use `recovery interval in minutes` to reduce the pressure on buffer washing, and to maintain replaceable buffers.The lower the value for `recovery interval in minutes`, the more frequently SAP ASE performs a checkpoint and washes all changed buffers. However, you must balance the benefits of performing the buffer wash with the number of disk I/Os that occur while SAP ASE is performing the checkpoint.

When you set `recovery interval in minutes` to higher values, you must balance the benefit of a smaller number of disk I/Os that result from less frequent checkpoints with the possibility that a request for a buffer may find it "dirty," delaying its use while the buffer wash takes place.

`recovery interval in minutes` applies to all databases server-wide, and you should change this parameter only after evaluating the impact this change has on full-durability databases. If you have at least one full-durability database that generally needs significant recovery after a server crash, continue to use the value required for a timely recovery of this database (generally, the default value of 5 minutes). If a change has little impact on full-durability databases (that is, they require little recovery after a server failure), start with a recovery interval higher than the default value of 5 minutes (for example, 30). After the change, view the Buffers Grabbed Dirty value or the per-cache information in the Data Cache Management sections of `sp_sysmon`. If the Buffer Grabbed Dirty value is high, decrease the value for `recovery interval in minutes`.

To decrease the number of I/Os to disk checkpoint performs for relaxed-durability databases without affecting the behavior of full-durability databases (regardless of the value for recovery interval), set the `no chkpt on recovery` database option to `true` for the relaxed-durability database. Use the method described above to evaluate the value for Buffers Grabbed Dirty and verify that disabling checkpoints does not negatively impact the availability of reusable buffers.

2.8.6 Minimally Logged DML

During minimally logged DML, `insert`, `update`, `delete`, and slow `bcp-in` commands are performed with minimal or no logging. If a statement fails (for example, from an internal error or a user-issued rollback), the portion of the work already completed stays committed, and is not rolled back.

However, when configured for minimally logged DML, SAP ASE must maintain logical consistency for changed tables (for example, rolling back a data row insert must result in a rollback of the related index rows). To ensure this consistency, minimally logged DML commands are divided into elementary unit operations named subcommands, which are all the data changes required to perform a single row change, including those for indexes and any text, image, or off-row columns. To maintain the logical consistency at the subcommand level, SAP ASE logs minimally logged DML commands in the user log cache (ULC). Once a subcommand completes, SAP ASE discards the log records for the subcommand from the ULC, there is no need to flush the ULC to `syslogs`.

If a ULC is not large enough to contain all of a subcommands' log records, SAP ASE may be unable to discard the log records. This generally happens if the data row affected by the DML is very large, or if a table contains many indexes. If SAP ASE cannot discard the ULC, it flushes the ULC log records to `syslogs`, which:

- Increases log contention in a busy system
- Impedes transaction throughput
- Increases the amount of log space required, which offsets any benefits gained by using minimally logged DMLs

Make sure that ULCs are large enough to contain the log records for most subcommands. SAP ASE recommends that your ULC be twice as large as the default server page size:

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "user log cache size", @ulc_size
```

For in-memory temporary databases or relaxed-durability temporary databases for which the durability has been explicitly set, improve the efficiency of minimal logging and avoid ULC flushes to `syslogs` by creating the sessions `tempdb`-specific ULC twice the size as the server logical page size:

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "session tempdb log cache size", @ulc_size
```

Generally, these scripts configure a ULC sized to contain all changes from one subcommand entirely in-memory, and yield significant improvement in concurrent DML performance.

Changing the ULC size requires you to restart the server.

To determine if minimal logging is efficient, view the Transaction Management section of `sp_sysmon` output.

This example shows efficient minimally logged DML, because SAP ASE discards most log records generated from subcommands:

```
ML-DMLs ULC Efficiency    per sec      per xact       count  % of total
------------------------  -------  ------------  ----------  ----------
Discarded Sub-Commands    33383.9       11087.8     3071323       100.0
Logged Sub-Commands           0.4           0.1          37         0.0
```

The Transaction Detail section presents a summary of DML commands performed in fully logged versus minimally logged mode for the specified sample. In this output, SAP ASE performed nearly all the inserts in minimally logged mode on a data-only locked table:

```
Transaction Detail            per sec      per xact     count  % of total
------------------------   ------------  ------------  -------   ----------
Inserts
  Fully Logged
   APL Heap Table                57.8         173.5      694        0.7 %
   APL Clustered Table            0.0           0.0        0        0.0 %
   Data Only Lock Table           0.7           2.0        8        0.0 %
   Fast Bulk Insert               0.0           0.0        0        0.0 %
  Minimally Logged
   APL Heap Table                 0.0           0.0        0        0.0 %
   APL Clustered Table            0.0           0.0        0        0.0 %
   Data Only Lock Table        7775.8       23327.5    93310       99.3 %
```

The Transaction management section provides details about how SAP ASE logs subcommands instead of discarding them. The output below shows the events causing ULC flushes to the transaction log, with a break-up of ULC flushes caused by fully logged and minimally logged DML. For the minimally logged DML section, nearly an equal amount of flushes were due to a Full ULC and to the end of a sub-command:

```
ULC Flushes to Xact Log         per sec      per xact     count  % of total
------------------------   ------------  ------------  -------   ----------
Any Logging Mode DMLs
by End Transaction                0.3           1.0        4       11.1 %
by Change of Database             0.0           0.0        0        0.0 %
by Unpin                          0.0           0.0        0        0.0 %
by Other                          0.0           0.0        0        0.0 %
Fully Logged DMLs
  by Full ULC                     0.2           0.5        2         5.6 %
  by Single Log Record            0.0           0.0        0         0.0 %
Minimally Logged DMLs
  by Full ULC                       1.3           4.0       16        44.4 %
  by Single Log Record              0.0           0.0        0         0.0 %
  by Start of Sub-Command           0.0           0.0        0         0.0 %
  by End of Sub-Command             1.2           3.5       14        38.9 %
------------------------     ------------  ------------  -------   ----------
 Total ULC Flushes                  3.0           9.0       36
```

If the value of the `count` column in the ULC Flushes to Xact Log by Full ULC section for Minimally Logged DMLs is high compared with the number of rows being affected, increase the value for the `user log cache` or the `session tempdb log cache size` configuration parameters.

The output below indicates the efficiency of ULC operations for minimally logged commands: the system incurs a small amount of logging overhead from minimal logging because nearly all the logging activity is entirely contained within the ULC, and very little flushing to `syslogs`.

```
ML-DMLs ULC Efficiency          per sec      per xact     count  % of total
------------------------   ------------  ------------  -------   ----------
Discarded Sub-Commands         7774.7       23324.0    93296        100.0
Logged Sub-Commands               1.2           3.5       14          0.0
-------------------------  ------------  ------------  -------
Total ML-DML Sub-Commands      7775.8       23327.5    93310
```

2.8.7 Dumping and Loading In-Memory Databases

Dumping and loading relaxed-durability databases is identical to dumping and loading full-durability databases.

During a dump of a relaxed-durability database, Backup Server reads directly from the disk-based database devices, During a load, Backup Server writes directly to these devices by copying over pages from the dump archive. You can improve dumping and loading performance by using striped devices for relaxed-durability databases.

Because in-memory databases do not use disk devices, Backup Server reads pages directly from SAP ASE shared memory during a dump, and writes it to the archive medium. During a load, Backup Server reads pages from the archive medium (for example, tape) and writes the load directly to the pages of the in-memory storage cache. Because in-memory databases do not use disk I/O to read or write pages on the server, the dump and load performance for in-memory databases is generally superior to that of disk-based databases of the same specifications.

During the `dump` and `load` commands, Backup Server opens one CT-library connection per stripe to SAP ASE, and creates a communication channel to read database pages directly from SAP ASE shared memory.

Backup Server synchronizes directly with concurrent tasks active in SAP ASE while reading pages directly from its shared memory. To support `load database` recovery, SAP ASE disables the strategies described in *Performance Optimization for Low-Durability Databases* for the duration of the `dump` operation. This may reduce the performance of transactional activity during the `dump database` command.

Related Information

Performance Optimization for Low-Durability Databases [page 38]

2.8.8 Tuning for Spinlock Contention and Network Connections

In-memory databases do not have as many latency and contention issues as disk-resident databases because they do not use disk I/O.

However, in heavy workload situations, in-memory databases may suffer from other spinlock contention and in-memory access issues:

- Cache spinlock contention – may become a bottleneck for in-memory caches under a heavy workload. Consider increasing the number of cache partitions to 64 or more. The additional memory resources required for large number of cache partitions is insignificant and provide improved performance by reducing the cache manager spinlock contention.
- Object manager spinlock contention – if your application frequently accesses a small number of objects, you may observe spinlock contention for metadata structures, which are reported by `sp_sysmon`. Bind descriptors for frequently accessed objects using `dbcc tune 'des_bind'` so they are never scavenged. Binding the descriptors for even a few commonly used objects may greatly reduce the overall metadata spinlock contention, and improve performance.

2.8.8.1 Improving Contention for Lock Manager Hashtable Spinlock Ratios

The throughput for in-memory databases may produce contention for the lock manager hashtable spinlock ratios. The table lock hashtable, and the page and row lock hashtable spinlocks may contribute considerably to the contention.

The Lock Management section of `sp_sysmon` shows the percentage of contention for the spinlocks that govern hash buckets for these hashtables:

```
Lock Management
---------------
Lock Summary                 per sec     per xact     count     % of total
-------------------------  ----------  ----------  --------    ----------
Total Lock Request          285063.3         43.0  17103795          n/a
Avg Lock Contention           1857.3          0.3    111435        0.7 %
Cluster Locks Retained           0.0          0.0         0        0.0 %
Deadlock Percentage              0.1          0.0         8        0.0 %
Lock Detail                  per sec     per xact     count     % of total
-------------------------  ----------  ----------  --------    ----------
Table Lock Hashtable
  Lookups                  130160.4         19.6   7809622          n/a
  Avg Chain Length             n/a          n/a   0.00000          n/a
  Spinlock Contention          n/a          n/a       n/a         4.6 %
[...]
Page & Row Lock HashTable
  Lookups                  268968.1         40.6  16138085          n/a
  Avg Chain Length             n/a          n/a   1.03330          n/a
  Spinlock Contention          n/a          n/a       n/a          4.8 %
```

Generally, if this contention is more than approximately 4 percent, consider reducing the ratio of these spinlocks to hash buckets. The default value for the configuration option controlling the number of hash buckets controlled by one spinlock is:

- `lock table spinlock ratio` with a default value of 20 affects the spinlock contention on the Table Lock Hashtable output.
- `lock spinlock ratio` with a default of 85 affects the spinlock contention on the Page & Row Lock Hashtable output.

When the spinlock contention is significant, reducing the values for `lock table spinlock ratio` and `lock spinlock ratio` may improve run-time performance. The additional memory overhead of more spinlocks controlling fewer hash buckets is not significant. Initially, reduce the configuration parameters by one half their values. In cases of extreme spinlock contention (in excess of 10 percent), reducing the appropriate configuration option to a very small value (say, .3 – 5), may help remove performance bottlenecks from spinlock overheads.

2.8.8.2 Determining the Number of Network Connections

By default, SAP ASE uses a single network listener, which can run on any engine.

When SAP ASE receives a connection request, the engine that accepts the connection becomes the network engine for that connection, and when the corresponding task performs network I/O, it must migrate to this engine.

If many client connections take place simultaneously, SAP ASE may not be able to schedule the listener between the connection requests (unless it yields due to running out of timeslice), and accepts all connections on the same engine. Since any of the corresponding tasks must run on this engine to perform network I/O, this engine becomes a bottleneck. Use the `sp_sysmon` Network I/O Management section to determine how SAP ASE distributes the network I/O. This example shows SAP ASE not distributing the network I/O proportionally: Engine 2 uses more than 88 percent of the network I/O, while other engines use as little as 0 percent:

```
Network I/O Management
----------------------
Total Network I/O Requests       7301.5           1.4      438092
n/a
Network I/Os Delayed              0.0           0.0           0
0.0 %
Total TDS Packets Received      per sec     per xact     count      % of total
--------------------------  ------------  ------------  --------  ------------
Engine 0                         308.8           0.1    18528          7.7 %
Engine 1                         163.6           0.0     9818          4.1 %
Engine 2                        3558.8           0.7   213527         88.3 %
Engine 3                           0.0           0.0        2          0.0 %
Engine 4                           0.0           0.0        0          0.0 %
Engine 5                           0.0           0.0        0          0.0 %
--------------------------  ------------      --------  -------
Total TDS Packets Rec'd          4031.3           0.8   241875
Avg Bytes Rec'd per Packet          n/            n/a      136
n/a
-------------------------------------------------------------------------------
Total TDS Packets Sent          per sec     per xact     count     % of total
--------------------------  ------------  ------------  --------  ------------
Engine 0                         308.8           0.1     18529          7.7 %
Engine 1                         163.6           0.0      9818          4.1 %
Engine 2                        3558.9           0.7    213531         88.3 %
Engine 3                           0.0           0.0         2          0.0 %
Engine 4                           0.0           0.0         0          0.0 %
Engine 5                           0.0           0.0         0          0.0 %
--------------------------  ------------  ------------  --------
Total TDS Packets Sent          4031.3           0.8    241880
```

To resolve unbalanced network I/O usage, use multiple network listeners and bind them to different engines (typically, one listener per engine). To determine how many clients to bind to each network listener, divide the client connections so that each listener accepts approximately the same number of connections. For example, if there are 6 network listeners and 60 clients, connect each group of 10 clients to one listener.

The `sp_sysmon` output after balancing the network listeners above looks similar to:

```
Network I/O Management
----------------------
Total Network I/O Requests       8666.5           1.3     519991            n/a
n/a
Network I/Os Delayed                0.0           0.0          0          0.0 %
Total TDS Packets Received      per sec     per xact     count      % of total
--------------------------  ------------  ------------  --------  ------------
Engine 0                         893.4           0.1     53602         17.8 %
Engine 1                         924.5           0.1     55468         18.5 %
Engine 2                         701.9           0.1     42113         14.0 %
Engine 3                         906.0           0.1     54358         18.1 %
Engine 4                         896.1           0.1     53763         17.9 %
Engine 5                         683.8           0.1     41028         13.7 %
--------------------------  ------------      --------  -------
Total TDS Packets Rec'd          5005.5           0.8   300332
Avg Bytes Rec'd per Packet          n/            n/a      136
n/a
-------------------------------------------------------------------------------
Total TDS Packets Sent          per sec     per xact     count     % of total
--------------------------  ------------  ------------  --------  ------------
```

```
Engine 0                          893.3          0.1     53595         17.8 %
Engine 1                          924.5          0.1     55467         18.5 %
Engine 2                          701.9          0.1     42113         14.0 %
Engine 3                          905.9          0.1     54355         18.1 %
Engine 4                          896.1          0.1     53763         17.9 %
Engine 5                          683.8          0.1     41026         13.7 %
------------------------   -----------  -----------  --------
Total TDS Packets Sent           4031.3          0.8    241880
```

Unbalanced network listeners are not specific to in-memory databases, but can also occur in disk-resident databases. However, because they do not use disk I/O, in-memory-resident databases typically have greater throughput than disk-resident databases. The increased throughput and non-existent disk I/O latency results in in-memory databases performing more work than disk-resident databases, including more network I/O, which could increase the severity of the bottleneck due to unbalanced network listener loads.

3 In-Memory Row Storage

The in-memory row storage cache stores transactionally active data, providing a high-performance storage repository for hot data (hot tables or hot partitions) and extending the table's traditional, page-based storage.

The in-memory row storage (IMRS) cache supports hot tables — the entire table is memory resident, and is constantly scanned or updated — and hot data, which is a portion of the table, usually small, that is transactionally active and is memory resident. The IMRS cache is an extension of the database device space allocated to the database, providing additional in-memory storage to store frequently accessed rows from the IMRS-enabled database.

Not all tables reside in the IMRS cache. Usually, only a small number of hot tables in an OLTP schema uses the IMRS cache (referred to as IMRS-enabled tables). The server inserts all new data into the IMRS cache, which it accesses and updates directly in-memory. Typically in transactional applications, once the server inserts a data record, it is accessed (and perhaps modified) for a short period of time immediately following the `insert` command. Usually, over time, the data that was once active is accessed or updated less frequently, and becomes passive data. Occasionally, passive data may be reactivated by regularly occurring business events (for example, end-of-quarter and end-of-year accounting, targeted sales promotions, and so on), at which time data becomes active again, residing in the IMRS cache. Over time, passive data becomes inactive, and is moved to a less efficient storage and access mechanism.

A disk-resident, fully durable database enabled for IMRS is a traditional database, supporting all the ACID properties of a regular disk-resident database, providing full support for `load database` and `load transaction` commands, while delivering improved performance for transactional workloads. Although an in-memory database is entirely in-memory, only a small portion of the IMRS-enabled database is in-memory. There are very few changes you need to make (potentially none) to existing applications for them to work with an IMRS.

You can create a new IMRS-enabled table, or alter an existing table to be IMRS cache-enabled. Only datarows-locked tables can be IMRS-enabled. Every IMRS-enabled table in the database is expected, but not required, to have a primary key index or an unique index. The rows for a given table in the IMRS are stored per-partition. You can also create and drop indexes on IMRS enabled tables that have index hash caching enabled.

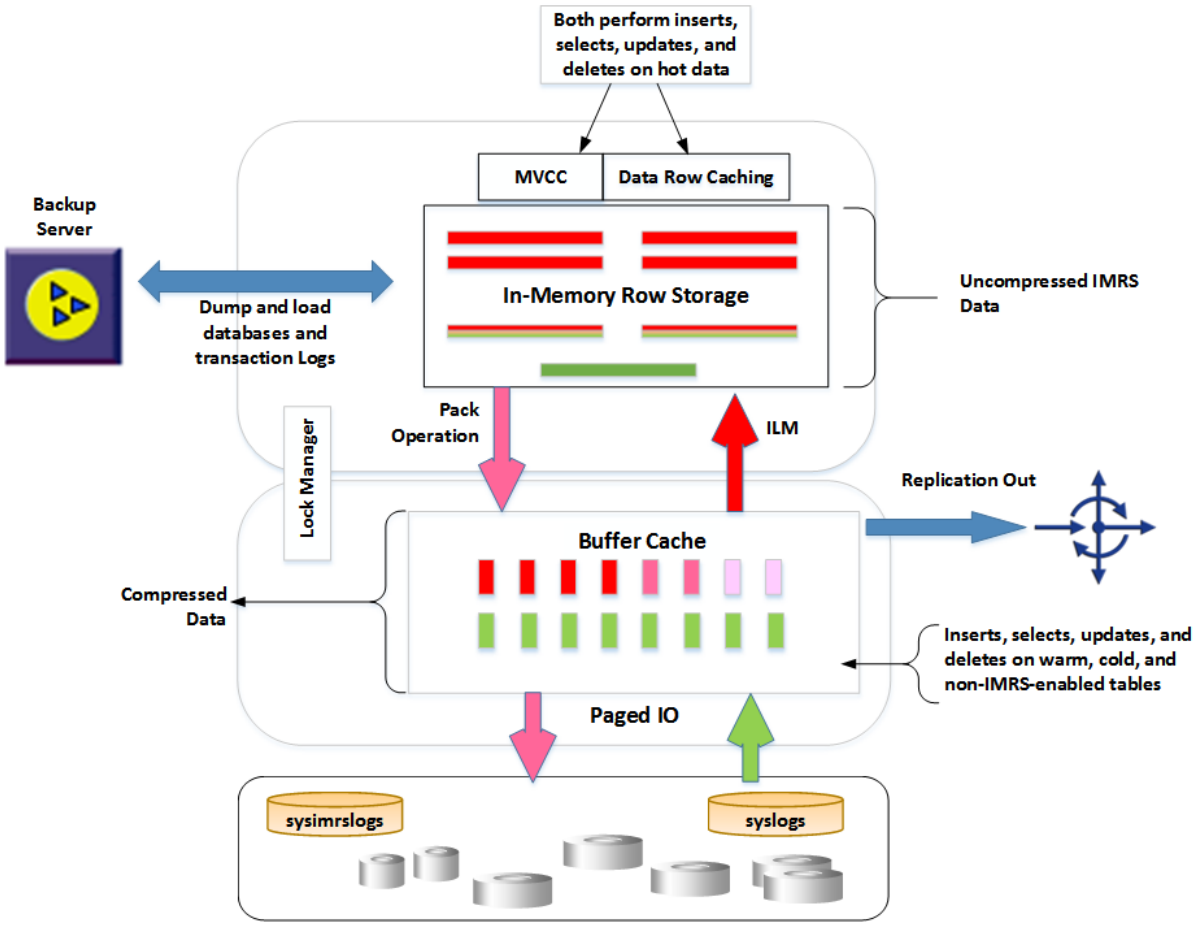
Multicolumn composite keys in the IMRS cache-enabled database allow all existing forms of indexes, including unique and non-unique indexes, local and global indexes, clustered indexes on datarows-locked tables, and so on. In-memory storage cache-enabled databases support all ANSI isolation levels, including `read committed snapshot` and `serializable snapshot`.

The IMRS infrastructure allows you to execute a single statement that affects some rows in the page or version store, and others in the IMRS. For example, a batch `update` statement affecting multiple rows may start by migrating an initial set of rows to the IMRS and then to avoid overwhelming the IMRS. Then the remaining rows continue to be updated in the page store.

The IMRS supports information lifecycle management (ILM) strategies. ILM constitutes a set of strategies and heuristics, using the IMRS-cache usage metrics to determine whether to relocate rows directly in the IMRS or to continue operating in the buffer cache or page store. At run-time, information is tracked to detect patterns of access to hot pages or hot rows. When hot data becomes less frequently used in the IMRS, it may be moved to the page store using an internal operation called pack, which moves older (cold) rows from the IMRS to the page store.

The IMRS pack is performed by the pack background process, which periodically scans the IMRS for memory usage and ILM metrics on row usage. Periodically, rows that meet the criteria are removed from the IMRS and moved to the page store, freeing the IMRS memory for new rows. Pack is a form of recycling unused (or infrequently used) rows from the row cache, similar to mechanisms that recycle buffers from the data or buffer cache. In the case of the buffer cache, recycling a dirty buffer may sometimes involve flushing it to disk to persist the contents before recycling. Similarly, pack writes the data rows to page storage so later on the dirty buffers can be flushed to disk to ensure persistence for the row data being evicted from the row cache.

The system architecture is described below:



The IMRS provides multiple in-memory storage strategies:

- Data Row Caching (DRC) – stores "hot" rows directly in-memory to deliver enhanced performance to run-time OLTP workloads.
- Multi-Version Concurrency Control (MVCC) – uses timestamp-based snapshot isolation that allows concurrent reads and writes.

3.1 On-Disk Row Storage Log

The IMRS is a cache for frequently scanned, cached rows, and a store for inserted rows and for new versions created by updates. Durability at commit-time and recovery for IMRS contents is provided by commit-time logging performed in the on-disk row storage (ODRS) log.

The ODRS log is a region of the database dedicated as the persistent storage for data contained and changed in the IMRS. The ODRS log is used for logging changes from committed transaction, and as the storage repository from which the contents of the IMRS are re-instantiated during a recovery operation.

The ODRS log is a page-oriented image of the contents of the row-oriented IMRS, and requires a similar amount of space (plus additional space to log overheads) to hold the contents. See Determining the Size of the IMRS Components [page 57] for sizing information.

The ODRS includes these entities as part of the database physical storage layer:

- Database devices of type `imrslog`, defined for this storage layer.
- A system-created named segment, `imrslogsegment`, identifying space allocated from the database devices to the ODRS.
- The `sysimrslogs` system table, which specifies the table-oriented layout for the space in these devices (similar to the way `syslogs` specifies the layout for the transaction log).

Commit-Time Logging

The server uses commit-time logging in `sysimrslogs` to track transactions that affect rows in the IMRS. When possible, the server accumulates all changes to an in-memory row and performs aggregated logging at commit-time. Runtime rollbacks are performed using in-memory data from the affected rows and any transactional changes. Change to the B-tree indexes are fully logged and are undone, or redone, using log records in `syslogs`.

Essentially, the commit-time-only logging and in-memory runtime rollbacks (which are not log based) result in the transaction log captured in the ODRS being a redo-only log, and undo logging is never seen in the ODRS log. If a transaction accesses regular and in-memory row storage tables, the server generates inline log records in `syslogs` and commit-time log records in `sysimrslogs`.

Processing commits in the IMRS requires both parts of the transaction to commit in `syslogs` and `sysimrslogs`; however, this is a simple concurrent `commit` operation that does not require sophisticated two-phase commit sequencing. An error to commit either side of the operation results in a transaction rollback.

> **i Note**
>
> Previous versions of SAP ASE did not perform a `group commit` for `sysimrslogs`. Versions 16.0 SP03 PL04 and later perform a `group commit` by default.

During a `checkpoint`, SAP ASE ignores `group commit` operations and flushes the changed `imrslog` buffers, even if more changes are expected on the same buffer. This occurs because, in exceptional situations, rogue transactions that reserve space and then hang can cause other transactions writing to the same `imrslog` page to hang due to the `group commit`.

3.2 Types of Rows Hosted by the IMRS

The IMRS includes inserted, migrated, and cached rows.

- Inserted rows – are newly inserted in an IMRS-enabled table. These rows are inserted directly in the IMRS, and do not include any page-resident images of the row in the table. Future updates to inserted rows continue to occur in the IMRS, but the row is still referred to as an "inserted" row.
- Migrated rows – reside in the page store but are migrated to the IMRS to support multiversion concurrency control (MVCC) or the datarow cache (DRC) features. If rows are migrated for updates or deletes, the contents of the row in the IMRS is the latest version, and the version of the row in the page store may be stale.
- Cached rows – reside in the page store and are frequently accessed and, consequently, are migrated to the IMRS without any changes to their contents. The contents of the cached rows in the IMRS and their original image in the page store are identical.

3.3 Interaction Between ILM and IMRS

Information lifecycle management (ILM) attempts to keep hot data in the IMRS.

ILM stores rows that are affected by inserts, selects, updates, and deletes in the IMRS if they become hot due to access patterns. The ILM uses the following rules to detect hot rows on IMRS-enabled tables. By default, these rules apply to all inserts, selects, updates, and deletes on DRC-enabled tables, but not to apply to DMLs on MVCC-enabled tables, since every DML on these tables must occur in the IMRS to provide row snapshots.

Rules affecting IMRS usage:

- Replication – ILM is disabled for DMLs in the IMRS if replication is enabled (that is, all DMLs are performed in the IMRS if replication is enabled).
- Cache usage – prevents the addition of new rows to the IMRS when the cache utilization is high. This helps prevent out-of-memory errors when performing operations in the IMRS and avoids performance issues when the system is running at high cache utilization. The server determines a threshold for IMRS cache utilization based on the `imrs cache utilization` configuration parameter. Above this threshold, it disallows inserts, selects, updates, and deletes to store rows in the IMRS.
- Scan pattern – uses the access path selected for a table. Rows are not migrated if a table scan is selected as an access path because subsequent table scans on rows stored in IMRS may be slower, and migrating rows to the IMRS due to table scans may flood the IMRS. If an index is selected as an access path, rows may be migrated based on the number of rows the scan accesses. For example, selects and updates migrate rows from the page store to the IMRS in point queries often seen in transactional systems that are driven by fully qualified index searches affecting only a single row.
- Number of rows brought to the IMRS – limits the number of rows brought into the IMRS by a single `insert`, `update`, or `select` statement. This rule limits a single statement from adding a large number of rows to the IMRS. Often, if a statement operates on numerous rows, not all of them are hot or are useful for subsequent statements. For example, an index scan that affects a small number of rows is a more reasonable candidate for hot rows than a range scan affecting 70% of a large table.
- Buffer hotness – captures the non-blocking advantage of IMRS over the concurrency issues experienced with buffer latching. If a buffer is frequently accessed or is updated concurrently, rows in the buffer can

avoid latching by using the IMRS. If the number of concurrent tasks accessing a buffer is above a specific threshold, rows in that buffer are considered hot and may be moved to the IMRS.

- Disabled partitions – the `sp_dboption ... auto imrs partition tuning` database option may disable the IMRS from being used for `insert`, `update`, or `delete` commands on partitions, avoiding the possibility of using the IMRS cache for cold data. For example, access to data in partitions older than 12 months in a time-based range partition table may be disabled from using the IMRS because these partitions are accessed infrequently.
  See *Reference Manual: System Procedures* for information about sp_dboption. See Automatic Partition Tuning [page 51] for information about auto-tuning workloads.

> **i Note**
>
> ILM rules do not apply to DML statements that require deferred mode processing. These statements always perform operations in the IMRS by migrating rows that are not already in the IMRS, and inserting new rows directly in the IMRS.

3.3.1 Automatic Partition Tuning

The server monitors workloads and identifies partitions of datarow-enabled tables suited for IMRS usage.

Tables containing archive data may not be suited to IMRS because they are unlikely to have frequent selects or updates. Without partition tuning, data for these partitions may be brought into the IMRS and subsequently packed without ever being reused, wasting CPU cycles and IMRS cache space.

> **i Note**
>
> The automatic partition tuning strategy is not applicable to MVCC-enabled tables.

The IMRS partition tuning feature is controlled by database option `auto imrs partition tuning`, which is enabled by default. When it is enabled, the server internally monitors the access patterns to rows in partitions, and disables or re-enables IMRS usage for various operations on the partitions. .

When `auto imrs partition tuning` is disabled, the server does not make any new decisions regarding IMRS partition usage. However, the server does continue to enforce existing partition tuning decisions. The tuning decisions are persistent across restarts of the server. Partition tuning is performed by a background thread that runs periodically.

Tuning is triggered internally after a number of transactions affecting the IMRS have completed. This tuning activity reacts dynamically to the changing workload pace; and the greater the transaction throughput, the more frequently tuning occurs.

Enabling or disabling IMRS-usage by a partition is done with respect to the row type (inserted, migrated, or cached). For example, in a list-partitioned table organized by state, new inserts should go to the IMRS for better performance, and infrequent accesses or updates to existing data in some states should not go to the IMRS. Consequently, partition tuning may choose to enable IMRS usage for inserted rows on a state (for example, a partition named `Calif`) because the latest data is considered hot, while disabling IMRS usage for infrequent selects (cached rows), or updates (migrated rows) on existing older data in the same partition.

Use `sp_imrs 'show'` to view this information. See the *Reference Manual: System Procedures*.

3.3.2 Determining the Number of Rows a Single Query Can Move to the IMRS

By default the number of rows a single query can move to the IMRS is 100. However, this value is configurable.

Rows that a query moves to the IMRS are called cached data rows, while those rows brought to the IMRS because they were accessed by a query are called cached access rows. To set the number of cached data rows, use the `create table` and `alter table` commands. The syntax is:

```
create table <table_name> with row_caching on row_caching {on [ allow [<number>
| default] ] | [for [ all | default ] ] rows ] | off}
alter table <table_name> set row_caching {on [ allow [<number> | default] ] |
[for [ all | default ] ] rows ] | off}
```

Where the maximum number of cached access rows is determined by:

- `<number>` – indicates the value for the upper limit for the number of cached data rows. The maximum value for the upper limit is 2147483646.
- `default` – (default behavior) for all subsequent queries, only the rows that are qualified and fulfill the criteria of the ILM rules are moved to the IMRS. For example, `alter table` and `create table` statements that include the `with allow` parameter use the default behavior.
- `all` – all accessed rows are cached for all subsequent queries.

The number of cached data rows—that is, the number of rows a query is moving to the IMRS—becomes the cached data rows threshold when you use the `allow` parameter to increase potential number of rows brought to the IMRS. Use the `for all` parameter to determine which rows are brought to the IMRS. This is not the default behavior for IMRS.

This example creates a table named `new_table` that allows for up to 1000 rows to be cached into the IMRS in a single query:

```
create table new_table (
    column1 int,
    column2 int)
lock datarows with row_caching on allow 1000 rows
```

This example alters the `pubs2.titles` table to allow up to 1000 rows to be cached to the IMRS in a single query:

```
alter table titles
set row_caching on allow 1000 rows
```

Once the `titles` table is configured, the processing of rows differs depending on how many rows the query moves:

- 15 rows or fewer – the rows are cached directly to the IMRS.
- Between 16 and 1000 rows – other ILM rules determine if these rows should be moved to the IMRS.
- 1000 or more rows – these rows are processed on the datapage.

Use the `alter table … default` parameter to reinstate the default value for the number of rows moved to the IMRS:

```
alter table titles
set row_caching on allow default rows
```

Use the `all` parameter to cache all the rows accessed by the query. These rows are cached when they are accessed. For example:

```
alter table titles
set row_caching on all rows
```

`sp_help` and `sp_help terse` output describe the current setting for the number of rows moved to the IMRS, and include information about `allow <number>` and `allow default`. For example, this output (which is shortened for space considerations) for the `titles` table includes information about the `cached data row thresholds` (see the emphasized lines):

```
sp_help titles
 Name   Owner Object_type Object_status
        Create_date
 ------ ----- ----------- ------------------------------------------
        -------------------
 titles dbo   user table  data row caching, cached data row threshold
        Aug 22 2018  3:16PM
. . .
 contract    bit           1 NULL  NULL     0              0 NULL
        NULL         NULL             NULL                          0
 attribute_class   attribute                int_value char_value comments
 ----------------- ------------------------ --------- ---------- --------
 misc table info   cached data row threshold      1000 NULL       NULL
Object has the following indexes
. . .
(1 row affected)
 concurrency_opt_threshold optimistic_index_lock dealloc_first_txtpg
        cached_index_root_page recompile_factor
 ------------------------- ---------------------- -------------------
        ---------------------- ---------------------------------------
                        15                     0                    0
                              0 NULL
(return status = 0)
```

The `sp_help` output indicates the change if you disable the `row_caching`, but still specifies that the table includes the `cached access rows`:

```
sp_help titles, terse
 Name     Owner      Object_type
Object_status

         Create_date
 -------- ---------- ----------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------- ---------------------------------------
 titles     dbo        user table             data row caching temporarily
disabled, cached accessed rows temporarily
disabled
             Jul 12 2018  2:15AM
```

If you configure the table for `row_caching for all rows`, `sp_help` output specifies that the `Object_status` is data row cached, but does not provide information about the attribute:

```
sp_help titles
 Name   Owner Object_type Object_status
        Create_date
 ------ ----- ----------- ---------------------------------------
        -------------------
 titles dbo   user table  data row caching, cached accessed rows
        Aug 27 2018  2:03PM
```

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

`sp_help terse` lists the cached access rows property as enabled:

```
sp_help titles, terse
 Name   Owner Object_type Create_date
 ------ ----- ----------- --------------------
 titles dbo   user table  Aug 27 2018  2:03PM
 . . .
 Table_property            Status/value
 ------------------------- --------------
 LOB compression level     0
 cached accessed rows      enabled
 cached_index_root_page    0
```

The values you set for cached data rows are persistent. That is, if you configure a table `for all rows` or `allow <number> rows` and then restart the server, the server moves the rows to the IMRS according to the parameters you set prior to shutting down the server.

If you perform the following sequence:

1. Configure a table `for all rows` or `allow <number> rows`.
2. Temporarily disable cached data rows with the `row_caching off` parameter.
3. Re-enable cached data rows with the `row_caching on` parameter.

The parameters you set in step 1 persist after step 3. That is, after step 3, executing any query moves rows to the IMRS according to the parameters you set in step 1.

3.4 Memory Requirements for IMRS

Using the IMRS is memory intensive. Because the IMRS runs entirely in-memory, it also assumes the system has a large amount of memory available.

The server allocates memory when connections first access IMRS- or MVCC-enabled tables so new statements need not allocate or free the memory. This memory is freed when the session is closed. In rare circumstances, when a high number of parallel user connections have accessed IMRS or MVCC-enabled tables, the server may issue error 725:

```
<SQL COMMAND>: failed to allocate 96 bytes from Adaptive Server's main memory
pool.".
```

SAP ASE uses IMRS memory for:

- RID-mapping table – a lockless table on each database where RIDs for rows residing in the IMRS can be mapped to the location of the in-memory row.
- Versions of data rows – In addition to data in each version of a row, the IMRS includes additional memory overheads for storing RIDs, timestamps, pointers to transaction structure, and older versions and column-oriented in-memory versions for cached rows.
- IMRS-related metadata information – several different metadata structures to track statements, transactions, garbage collection and pack thread operations, cached rows, partition-specific IMRS-metadata, run-time information on older versions and LOB versions, and so on.

Since the actual memory required depends on the number of concurrent transactions, data row and LOB versions created by such transaction, and recycling is limited by the oldest active transaction and scan, this value is close to the size required for the transaction log. You should configure this cache to about 2.5 times that of the transaction log size, with a minimum size of 2 GB, considering the additional memory overhead per version per data row.

3.5 Using the Garbage Collector to Reclaim Memory

Each `insert`, `update`, or `delete` on IMRS-enabled tables creates a new version of the data row in the IMRS or the version store.

Using a timestamp-based mechanism, the IMRS garbage collector reclaims all the row versions created before the oldest active transaction started in the database affecting IMRS-enabled tables. This ensures that older versions needed by snapshot isolation scanners are available in the system.

The oldest active transaction in the system affects which data row versions the IMRS garbage collector can free. Transactions or query scans using snapshot isolation see a snapshot view of the data based on timestamp they acquired when they started.

Long-running, and inactive transactions are preempted by the IMRS garbage collector, enabling the garbage collector to clear unused versions from the IMRS memory.

If a command running at the `statement snapshot` isolation level remains idle for more than 60 seconds, the IMRS garbage collector preempts the statement, and moves forward to the next statement.

If a command running at the `transaction snapshot` isolation level is idle for more than the time specified by the `snapshot isolation timeout` configuration parameter, the statement rolls back, along with the transaction in which it was executed. To indicate this rollback, the following error message is issued to standard out:

```
Error message 12357, "Command aborted because the configuration option 'snapshot
isolation timeout'
is set to <x> seconds, and the command execution is idle since <y> seconds."
```

Active scans and transactions view the versions of the data rows, and versions that have lower timestamps are discarded, and the memory they occupy is reused.

The IMRS garbage collection task wakes up and scrutinizes the list of all completed transactions. It determines which versions of completed transactions can be cleaned up, and reclaims the versions they create based on the `commit` timestamp of the version and the timestamp of the oldest active statement or transaction in the database.

There can be multiple `IMRS-GC` threads operating in a database, ensuring that memory reclamation is performed expeditiously. Garbage collection activity is performed in a non-blocking manner, and does not affect runtime transaction performance.

The IMRS cache may run out of memory if a session performs a `begin tran` and keeps the transaction open for a long time, or for a long-running statement, executes against IMRS-enabled tables. The IMRS garbage collector is also responsible for reclaiming memory allocated to various metadata structures (for example, RIDMAP table components, transaction descriptors, and so on). If you are using on-disk MVCC, the garbage collector frees space from `sysversions` in the version store.

This may occur if there are a sufficient number of configured GC threads, but they are blocked due to an old transaction and not able to progress. In this situation, `sp_who` shows that the IMRS GC threads are always sleeping.

You can also check the `WaitStatus` column of `sp_imrs show, gcinfo` or `sp_imrs clear, blocking_spid` to see if IMRS GC threads are blocked on any particular task. The following example shows that the `imrsdb` and `imrsdb2` databases are blocked by SPIDs 36 and 37, respectively:

```
sp_imrs 'show', 'blocking_spid'
 SPID UserName HostName    OSPID StartTime           NTransBlocked DBName  Status
 ---- -------- ----------- ----- ------------------- ------------- -------
----------
   36 dbo      linuxstore3 32441 Mar 29 2017  2:24AM             0 imrsdb  recv
sleep
   37 dbo      linuxstore3 6381  Mar 29 2017  2:27AM             0 imrsdb2 recv
sleep

To unblock GC of database imrsdb, execute the following -
sp_imrs clear, 'blocking_spid', 'imrsdb','36'
```

The information can also be filtered for a specific database:

```
sp_imrs 'show', 'blocking_spid', 'imrsdb2'
 SPID UserName HostName    OSPID StartTime           NTransBlocked DBName  Status
 ---- -------- ----------- ----- ------------------- ------------- -------
----------
   37 dbo      linuxstore3 6381  Mar 29 2017  2:27AM             0 imrsdb2 recv
sleep

To unblock GC of database imrsdb2, execute the following -
sp_imrs clear, 'blocking_spid', 'imrsdb2','37'
```

Use the `clear` parameter with `sp_imrs` to remove the blocking process:

```
sp_imrs clear, 'blocking_spid', 'imrsdb','38'
Successfully cleared Blocking SPID 38 from the system.
```

Issue `sp_imrs` periodically without specifying the spid to clear any processes that might be blocking the IMRS GC. In the following example, the database `imrsdb` contains a blocking SPID 38:

```
sp_imrs clear, blocking_spid, 'imrsdb'
Successfully cleared Blocking SPID 38 from the system.
```

`sp_imrs` returns silently if there are no processes to be cleaned.

3.5.1 Statement Registration and the Garbage Collector

The garbage collector cannot clear a version if a scanner is active on it. Each scanner registers itself with the garbage collector to make sure that no version is cleared after the scanner starts.

The IMRS maintains the oldest active statement (scanner) of the database. The garbage collector does not process any version that is created after the oldest active statement.

However, when a command (for example, `reorg rebuild` in offline mode) takes an EX_TAB lock on an object, it specifies the following:

- There are no older scanners active on the object.
- The utility is working on the latest version.
- The IMRS pack operation cannot create a new version because the command has an EX_TAB lock.

Because the garbage collector does not have anything to collect in this object after the command starts, there is no benefit in registering this command with the garbage collector. This is true for any command that does not interfere with the IMRS content (for example, `reorg reclaim space`, `reorg rebuild index`, and so on). If this command registers itself, the garbage collector is unnecessarily blocked for other objects in this database because the oldest active statement is maintained on database level.

However, there are some commands that can run inside a transaction if the `ddl in tran` database option is enabled —typically these are commands that do not require a data copy. Even if commands take an EX_TAB lock, registration is prohibited for them only if they are not running inside a transaction.

Related Information

Allow Data Definition Commands in Transactions

3.6 Determining the Size of the IMRS Components

An important step when creating the IMRS is determining the size of the components.

Determine the following:

- The size of the row storage cache.
- The size of on-disk row storage (ODRS) log (this includes `imrslogsegment` and size of devices of type `imrslog`).
- Which tables are best suited for datarow caching and for multi-version concurrency control (MVCC).
- Which indexes are best suited for use of latch-free B-trees indexes. See the *Performance and Tuning: Locking and Concurrency Control* for information about latch-free B-tree indexes.
- The resource requirement for the server when enabling one or more of these features on existing tables.
- The configuration changes required to effectively use these features for optimal performance.

Generally, the size of the IMRS is dictated by:

- Whether row storage is enabled database-wide, and if so how many tables the database contains.
- If row storage is enabled for select tables, the size of the tables.
- Whether IMRS will be used for datarow caching or for in-memory versioning to support MVCC.
  - If MVCC is enabled, the number of versions of a row, on an average, that is retained simultaneously in the cache.
  - If the MVCC-enabled or DRC-enabled tables have large in-row LOB columns, how frequently the MVCC-enabled tables (non-LOB and in-row LOB columns) are updated.

The size of `sysimrslogs` is determined by:

- The size of the IMRS; by the number, size, update frequency, and so on of IMRS-enabled tables.

- The imrslog log device is similar to the regular transaction log and must be larger (for example, 5 times larger) than the cache size.
- How frequently the IMRS-enabled tables are updated, including the frequency of the following:
  - The average size of inserts
  - The average size of updates (that is, of new rows generated by updates)
  - Deletes
  - Commits for transactions affecting IMRS entities

Use `sp_logiosize` to change the size of the log I/O for `sysimrslogs`. For example, this changes the log I/O size to 32K:

```
sp_logiosize "sysimrslogs" "32"
```

Use the `sp_wlprofiler` to determine starting recommendations for sizing. See Managing the Workload with sp_wlprofiler [page 72] and `sp_wlprofiler` in the *Reference Manual: Procedures*.

3.7 Configuring SAP ASE for In-Memory Row Storage

SAP ASE uses row caches to support IMRS.

Context

A row storage cache functions much like a page-based buffer cache. A row storage cache is not created for a specific table: The same row storage cache can be shared for data rows from multiple tables, with widely varying schemas. The amount of memory allocated for a table can vary according to variable-length columns, in-row LOBs (which can also be memory resident), expanding or shrinking updates, rollbacks, and in-memory versioning for snapshot isolation.

The kernel memory manager supports the row-storage caches, providing low-latency and non-blocking concurrent access to cache memory. The IMRS memory manager provides optimal and high-performance row memory management.

The IMRS is created as a named cache of type `row_storage`, implying that the memory from this cache is used for storing data as rows rather than in pages or buffers, as is normally done for a named cache or for `inmemory_storage` caches. The server uses the memory allocated to this cache only to create row buffers that are issued to individual transactions working on rows stored in the IMRS.

To use IMRS, first enable the `enable mem scale` configuration parameter, then enable the "`enable in-memory row storage`" configuration parameter.

Procedure

1. Enable SAP ASE for IMRS:

```
sp_configure 'enable mem scale', 1
go
sp_configure 'enable in-memory row storage', 1
go
```

2. If necessary, configure the garbage collection tasks and pack tasks for your site:

```
sp_configure 'number of pack tasks per db', <value>
go
sp_configure 'number of imrs gc tasks per db', <value>
go
sp_configure 'Number of lob gc tasks per db', <value>
go
```

   See the *Reference Manual: Configuration Parameters* for more information.

3. Use `disk init` to create an imrslog device for the cache. For example:

```
disk init name = 'pubs2_imrslog_dev',
physname = '/usr/u/sybase/devices/pubs2_imrslog_dev.dat',
size = '200g',
type = imrslog
```

4. Use the `sp_cachconfig ... row_storage` parameter to create the row storage cache. For example, this creates a 20 GB row storage cache named `pubs2_row_cache`:

```
sp_cacheconfig "pubs2_row_cache", "20g", "row_storage"
```

   You can also create databases that use row storage cache and alter existing databases to use IMRS. See:
   - Creating Databases that Use a Row Storage Cache [page 66]
   - Altering Database to Use In-Memory Row Storage [page 67]

> **i Note**
>
> Each cachlet in an in-memory cache must contain a full allocation unit of 256 pages. If the amount of space you request does not completely full all the allocation units, the server reduces the amount of space allocated until all remaining cachlets are full. The page size of the server determines the amount of space reduced. For example, on a server using 2K logical pages, `sp_cacheconfig` allocates 96MB of 100MB requested:

```
 sp_cacheconfig tempdb_user_cache, "100M", "inmemory_storage",
"cache_partition=32"
go
sp_cacheconfig tempdb_user_cache
go
 Cache Name                          Status
         Type                                Config Value
         Run Value
 ------------------------------- ----------------
         --------------------------------- ----------------------------
         ------------------------
 tempdb_user_cache                   Active
         In-Memory Storage                      100.00 Mb
             96.00 Mb
(1 row affected)
                                             ------------ ------------
```

```
                            Total     100.00 Mb     96.00 Mb
--------------------------------------------------------------------------------
Cache: tempdb_user_cache,   Status: Active,   Type: In-Memory Storage
      Config Size: 100.00 Mb,   Run Size: 96.00 Mb
      Config Replacement: none,   Run Replacement: none
      Config Partition:           32,   Run Partition:           32
 IO Size          Wash Size                  Config Size
         Run Size                 APF Percent
 ---------------- ------------------------- ------------------------
         ------------------------ ------------------
     2 Kb                    0 Kb                  96.00 Mb
             96.00 Mb                  0
(return status = 0)
```

However, on a server that uses 8K logical pages, `sp_cacheconfig` allocates 64MB of the 100MB requested:

```
 sp_cacheconfig tempdb_user_cache, "100M", "inmemory_storage",
"cache_partition=32"
go
sp_cacheconfig tempdb_user_cache
go
 Cache Name                          Status
         Type                                Config Value
         Run Value
 ---------------------------------- ----------------
         ---------------------------------- ----------------------------
         ------------------------
 tempdb_user_cache                  Active
         In-Memory Storage                     100.00 Mb
             64.00 Mb
(1 row affected)
                                               ------------ ------------
                            Total     100.00 Mb     64.00 Mb
================================================================================
Cache: tempdb_user_cache,   Status: Active,   Type: In-Memory Storage
      Config Size: 100.00 Mb,   Run Size: 64.00 Mb
      Config Replacement: none,   Run Replacement: none
      Config Partition:           32,   Run Partition:           32
 IO Size          Wash Size                  Config Size
         Run Size                 APF Percent
 ---------------- ------------------------- ------------------------
         ------------------------ ------------------
     8 Kb                    0 Kb                  64.00 Mb
             64.00 Mb                  0
(return status = 0)
```

Consequently, if you issue a command to create a 100MB cache: For this issue to show up, it requires larger page size and large cache partition values. in this example, for an 8K server, 2M of space is needed for an allocation unit of 256 pages. With 32 cache partitions, that means that the the inmemory cache has to be divisible by 64 (32 x 2). Hence the 100M is rounded down to 64M.

3.8 Data Row Caching

The in-memory row storage uses data row caching (as opposed to the page caching used in regular tables) to improve performance while scanning or updating frequently accessed rows.

Data row caching allows in-memory row storage to store data from frequently accessed or modified rows (known as “hot” rows). Rows are first inserted into an in-memory row storage-enabled table, not the page

storage. Subsequent access to those rows for selects or modifications occurs in the in-memory row storage, instead of the slower page storage, until the row is not longer considered to be "hot."

Use `create table` or `alter table` to enable a table for data row caching.

The name of the row storage cache assigned to the database is stored in the `imrscache` column of `sysdatabases`.

Databases with row storage need one or more devices of type `imrslog`. All `imrslog` devices are bound to `imrslogsegment`, the storage repository for the in-memory row storage log. Configure the space in `imrslogsegment` according to the size of the row storage cache defined for the in-memory row storage for the database.

Generally, enabling data row caching for a table includes these operations:

- New inserts go directly to the in-memory row storage without allocating any space in the page store. The in-memory row storage performs subsequent updates to these rows. The rows are deleted directly from the in-memory row storage, without requiring access to the page store.
- Frequent updates to rows residing in the page store may result in the server migrating the row to the in-memory row storage in an uncompressed format. Subsequent updates are performed to the version of the row in the in-memory row storage, without accessing the data page or data buffers.
- Depending on the access pattern to hot data, some frequently accessed rows may be cached in the in-memory row storage, in an uncompressed format. Subsequent scans are performed to the version of the row in the in-memory row storage, without accessing the data page or data buffers.
- Newly inserted or migrated rows residing in the in-memory row storage are fully accessible using the existing B-tree indexes.
- Once access to rows in the in-memory row storage decreases, a background task harvests old or cold data rows and moves them to the page store, releasing any memory used by these rows back to the in-memory row storage. Rows that were newly inserted to the in-memory row storage are moved to newly allocated pages. Existing rows that were migrated to the in-memory row storage from the page store are migrated back to their original home row location in the page store.

This example alters the `pubs2.returns` table for row storage:

```
alter table returns
set row_caching on
```

3.8.1 Row Caching Strategies

Caching rows in the IMRS is a performance accelerator strategy but can be overridden because of a temporary lack of available memory for rows in the IMRS, or when memory usage exceeds internally established thresholds on the IMRS.

The inability to cache frequently accessed rows, or any row from a hot table, is not an error condition, but may merely result in reduced performance.

The data in cached rows is identical to their original version that resides on the page store. Once you enable row caching for compressed data, the cached row is maintained in an uncompressed format, even though the original row that remains on the page is compressed. Once a row from the page-store is cached in the IMRS, there is no restriction that the associated data buffer remain in the buffer cache. The buffer could be recycled and used for another page, freeing some memory to the buffer cache. As long as the cached row remains in the

IMRS, it is accessed from the IMRS without any further access to the page store. Existing table scans and B-tree index scans work identically whether the row is an inserted or cached row in the IMRS, or resides on the page store.

Depending on the need for additional IMRS memory from other concurrent requests, the memory for cached rows may be periodically reclaimed and released back to the IMRS. Subsequent access to these cached rows from the index is redirected back to the rows' original page, which may require that the data page be brought back into cache from disk. The memory reclamation strategies attempt to retain cached rows from "hot" tables for a longer period of time (subject to other demands for memory from competing tasks). In general, how frequently a cached row is accessed in the IMRS determines how often they may be recycled and their memory reused for other rows.

Row caching is an unlogged operation, and cached rows are not captured in `sysimrslogs`, and are not contained in the IMRS archives. Cached rows are not reinstantiated or recovered from archives after a server restart. New rows may be cached into the IMRS over time, depending on their work load and access patterns.

During a rollback of a transaction or a rollback of a savepoint, any rows cached during the transaction that rolled back may remain cached in the IMRS after the rollback due to the unlogged nature of the row caching strategy. If the rows are needed again, they are returned directly from the IMRS. If the rows are not accessed, they are aged out of the IMRS and their memory reclaimed for other purposes.

3.9 Inserts, Selects, Updates, and Deletes in an IMRS-enabled Database

How inserts, selects, updates, and deletes behave depends on whether you have enabled the data-row caching, mulitversion concurrency control, or both.

The table below describes the general behavior of DMLs affecting single rows. Many run-time considerations (for example IMRS cache memory, ILM rules, and so on) may influence whether the DML operation is performed on the data row in the page store or in the IMRS.

Command	Data-Row Caching Enabled	Multiversion Concurrency Control Enabled	Both Enabled
`insert`	Inserts rows in the in-memory row storage. Subsequent selects, updates, and deletes to these rows are done directly in the in-memory row storage, without affecting the page store. ILM rules apply.	Updates rows in the in-memory row storage. Subsequent updates and deletes to these rows are done directly in the in-memory row storage, without affecting the page store.	Same behavior.

Command	Data-Row Caching Enabled	Multiversion Concurrency Control Enabled	Both Enabled
`select`	Hot rows that are frequently scanned are sent to the in-memory row storage for row caching. Subsequent selects of these rows return the values directly from the in-memory row storage. ILM rules apply.	Rows are returned from the page store, including frequently scanned rows not previously updated. Inserted and updated rows are returned from the IMRS.	Hot rows are sent to the in-memory row storage for row caching.
`update`	Performs updates to page store rows directly on the page store. Frequently updated rows are sent to the in-memory row storage. Subsequent updates to rows in the in-memory row storage are performed there, without affecting the data page. ILM rules apply.	Every update to a row residing in the page store also migrates a row to the in-memory row storage. Updates to rows in the in-memory row storage are performed there. Every `update` produces a new in-memory version of the row.	Same behavior.
`delete`	Performs deletes on the page store if the `delete` occurs there first. Deletes of frequently scanned or updated rows migrated to the in-memory row storage are performed there and in the data page. ILM rules apply.	Rows are returned from the page store, including frequently scanned rows. Every delete of a row residing in the page store also migrates a row to the in-memory row storage. Deletes of rows in the in-memory row storage are performed there. Every `delete` produces a new, but empty, in-memory version of the row	Same behavior.

See Multiversion Concurrency Control [page 101].

3.9.1 Additional Information on Issuing DMLs in an IMRS-Enabled Database

Some aspects of the `insert`, `update`, and `delete` commands function differently in the IMRS.

Inserts

All single-row inserts to an IMRS-enabled table, except those from a `bcp-in`, insert rows directly into the row storage cache, and do not perform any page allocation or access the data pages for data rows. Multirow inserts specified by syntax similar to `insert ... select from ...` insert some rows directly into the IMRS (subject to the number of rows that qualify based on ILM rules), while the remaining rows are inserted directly to the page store.

Any existing page-based indexes are updated while inserting rows directly into the IMRS. Unique and primary key constraints are enforced by using the defined index-key constraints (as they are for regular tables). Similar to regular tables, you can use rules, check constraints, and default values to insert data into IMRS-enabled tables.

Although you are inserting data into an IMRS-enabled database, the server continues to enforce existing row-size limits, data-only start offsets, and other limits based on its page size. These limits are enforced in anticipation of the row eventually being persisted to the page store, at which time inserting a row that was previously committed in the IMRS, but too wide for the page storage, may run into a physical limitation. Any violations result in the insert failing.

In-row LOB columns are fully supported while inserting new rows.

Inserts to the IMRS are fully logged operations with commit-logging performed in `sysimrslogs`. Other than reclaiming the memory, which is performed later by the background IMRS garbage collection thread, rolling back inserts to the IMRS requires no logging and is done fully in-memory.

Issuing `insert...select...use ins_by_bulk` into an IMRS-enabled table ignores the IMRS, and instead inserts directly to the page store, leveraging the performance capabilities of bulk copy and parallel index updates.

Selects

Frequently selected rows that reside on the page store may be moved and cached to the IMRS. The factors for determining whether to cache frequently accessed rows in the IMRS depends on internally tracked metrics such as, how frequently rows are accessed, and how "hot" a data page is.

IMRS-enabled databases support table scans for rows in the IMRS and regular B-tree-index as an access method. However, table scans are typically used for internal scans for utilities. The IMRS also supports range scans using the B-tree index and RID scans to support cursors. Typically, fully qualified selects in transactional workloads return a single row (in some cases, a small number of rows are returned): These selects on IMRS-enabled tables use index scans. When possible, a fully qualified index scan using a unique index returning a single row may be executed using a hash index defined on the index key columns of the B-tree index.

Updates

Updates to rows on the page store are performed in the page store. To improve performance for future selects or updates to these rows, the server applies runtime heuristics to identify rows in the page store that are frequently updated, and these rows are relocated to the IMRS for datarow column-enabled tables. This process occurs seamlessly, and is referred to as row migration. For example, single-row updates driven by a fully qualified unique index generally result in the row being moved to the IMRS, followed by an `update` (subject to memory availability in the IMRS cache).

Row migration is a non-logged operation in `sysimrslogs`. Because migrated rows may be re-accessed (they were initially tagged as frequently updated "hot" rows), rollbacks of the transaction that caused the row migration rolls back the `update` itself. If the update that migrated the row is rolled back, the migrated version is rolled back as well. That is, the update event may be a combination of an unlogged row-caching operation followed by a logged update operation.

Optimizations are provided to perform row migration without updates to indexes that point to the migrated row. The RID of the migrated row does not change, and the index scans are redirected to access the row from the IMRS by looking up the RID mapping table.

Updates to rows in the IMRS (that is, inserted, cached, or migrated rows) are performed directly in the IMRS, and do not affect their home-row location. Every update to rows in the IMRS produces new row-images, causing memory consumption while the transaction is active. The IMRS garbage collection background thread reclaims memory for intermediate row versions after the transaction completes.

For migrated and cached rows that are frequently updated in the IMRS, the latest version of the row is in the IMRS, but the original version in the page store becomes stale over time. Indexes that access these rows find the latest version of the row in the IMRS. If the row was previously migrated to the IMRS, table scans accessing rows from the page store return the latest version from the IMRS.

Some of the updated rows may be returned to the page store and deleted from the IMRS due to a decrease in the update frequency, a reduction in memory available to the IMRS, and so on.

Deletes

Deleting inserted rows from a row cache is performed as a logged-delete operation, and is logged in `sysimrslogs`. All memory used by the row and any of its intermediate versions is made available for the IMRS garbage collection thread to reclaim, after the transaction completes.

Deleting cached rows is performed as a logged-delete operation in the page store, and is logged in `syslogs`. This operation erases the footprint of the cached row in the IMRS, and the memory used by the row is made available for the IMRS garbage collection thread to reclaim, without logging the operation in `sysimrslogs` (logging to `syslogs` is performed the same as it's done for a regular delete). Reclaiming the memory after the delete is deferred until after the transaction completes so runtime transactional performance is not affected. This is beneficial also in case the deleted transaction needs to be rolled back: the in-memory version of the row may be safely re-instated without needing additional memory allocated or initialized.

Deleting migrated rows is slightly more complicated. The in-memory component of the migrated row is deleted the same as for inserted rows. However, because the migrated row includes a page-resident row-image, deleting the row from the page store is performed as a fully logged operation in `syslogs`. Logging the delete for the page-store version of the row occurs in-line with the transactional activity, but logging the in-memory

version is performed as a commit-time logging operation in `sysimrslogs`. During a rollback, the page-image version of the row is restored to its pre-updated image. The final version of the deleted row is reinstated in the IMRS, and does not require a memory allocation operation. Memory for all versions of the row is made available to the IMRS garbage collection thread to reclaim after the transaction completes.

Merges

A `merge` statement consists of two components: updating existing rows that qualify for the statement and, if they do not, inserting new rows. The strategies described in previous sections apply consistently to the update and insert components of a `merge` statement.

A single `merge` statement can consist of inserting some rows into the IMRS, and updating other rows in the IMRS and in the page store. If row-usage thresholds are met, some updates under a single `merge` statement could cause row migration, which works as described earlier. All of these events are possible, dictated by their row-use metrics, and by the rows that qualify.

Logging and rollback semantics of the `merge` statement is the same as for the individual insert and update components.

3.10 Creating Databases that Use a Row Storage Cache

You can dedicate an in-memory row storage cache to the database for tables eligible for in-memory row storage when you create a fully durable, disk-resident database.

Every database created with a row storage cache must also be created with the on-disk row storage device of type imrslog (this is the device space to create `sysimrslogs`).

Use the `create database` command to create databases that use row storage cache. See the Reference Manual: Commands.

The following example creates a database named `pubs3` database on a number of database devices, and newly created devices of type imrslog. Enable the use of an existing row storage cache, `prod_row_cache`, for this database:

```
create database pubs3
on data_dev1 = '500m', data_dev2 = '500m'
log on log_dev1 = '200m', log_dev2 = '150m'
imrslog on prod_odrs_dev1 = '200m',
prod_odrs_dev2 = '100m'
row storage on prod_row_cache
```

This cache is now available for tables created in this database to be used for row caching or snapshot isolation:

When you create a database with row storage, it needs one or more devices of type `imrslog` available. All `imrslog` devices are bound to the `imrslogsegment`, an internally created segment that is the storage repository for the on-disk row storage log. The amount of space for an adequately configured `imrslogsegment` depends on the size of the row_storage cache defined for the in-memory row storage for the database.

You need not immediately use the log space. You can create a database using the `imrslog on` clause to provision space in `sysimrslogs` for future use. When the log space is needed, issue `alter database` to use the row storage cache for this database. However, you cannot perform these steps in the reverse order (that is, creating a database with the `row_storage on` clause followed by an `alter database ... imrslog on` clause). Enabling row storage caches for the database may result in the server using the in-memory row storage for DMLs, which require the space in sysimrslogs at run time.

You can enable row caching and snapshot isolation with the `create database` command. Once enabled, all newly created tables automatically inherit row caching or snapshot isolation.

For example, this enables row caching on the `pubs3` database:

```
create database pubs3
on data_dev1 = '500m', data_dev2 = '500m'
log on log_dev1 = '200m',
imrslog on prod_odrs_dev1 = '200m',
prod_odrs_dev2 = '100m'
row storage on prod_row_cache
with row_caching on
```

The following creates the `pubs3` database with snapshot isolation:

```
create database pubs3
on data_dev1 = '500m', data_dev2 = '500m'
log on log_dev1 = '200m',
imrslog on prod_odrs_dev1 = '200m',
prod_odrs_dev2 = '100m'
row storage on prod_row_cache
with snapshot_isolation on
```

See the *Reference Manual: Commands* for the full syntax.

3.11 Altering Database to Use In-Memory Row Storage

Use `alter database` to configure existing disk-resident, full durability disk-resident databases to use row storage caches for newly created tables, to extend the databases to devices of type `imrslog`, to create space for the on-disk row storage (ODRS) log, and to enable or disable row caching or snapshot isolation database-wide.

Context

However, altering the database to use a row storage cache requires that you provide sufficient device space for the ODRS log. This may already exist, or you can include the additional space as part of the `alter database` command.

You can also use the `alter database` command to enable or disable row caching or snapshot isolation for all existing user tables in a user database.

Procedure

1. Create server-wide resources such as the IMRS cache and devices of type imrslog (see Configuring SAP ASE for In-Memory Row Storage [page 58].
2. Extend the `pubs2` database to use the new devices for the ODRS

   ```
   alter database pubs2
   imrslog on pubs2_imrslog_dev = '200M'
   ```

3. Create a named cache for row storage usage. For example:

   ```
   sp_cacheconfig "new_pubs2_row_cache", "50G", row_storage
   ```

4. Alter the database to use the row storage cache, providing the newly created row storage cache as the IMRS:

   ```
   alter database pubs2 row storage on new_pubs2_row_cache
   ```

   You can also use the `alter database` command to:
   - Enable database wide row caching for newly created (and existing) datarows-locked tables. For example, this enables row caching for all the existing, and future, tables in the `pubs2` database:

     ```
     alter database pubs2
     set row_caching on for all tables
     ```

   - Change the snapshot isolation level for the database with the `set snapshot_isolation` parameter. For example, this enables snapshot isolation database-wide on the `pubs2` database, so that, by default, all newly created datarows locked tables in `pubs2` are enabled for snapshot isolation:

     ```
     alter database pubs2
     set snapshot_isolation on
     ```

3.12 Creating and Altering Tables to Use Row Storage Cache

Use the `create table` command to create tables that use row storage cache.

Individual tables or partitions that are heavily updated or selected during transactional workloads can be created to use the IMRS. Bind the database to and enable it for row storage before creating a table to use the IMRS.

A table can span both the page store and the row store. The rows of the table move between the page store and row store, depending on the access patterns. This movement is driven by the ILM and the size of the row storage cache. Since this is a limited resource, use care to select which tables are enabled for row caching based on their access patterns in your applications.

Enable the database to use row storage cache before you create tables that use row caching or snapshot isolation. Use the `alter database` to configure existing databases to use a row storage cache. Once the database is enabled, individual tables use the cache as their in-memory row storage cache. The specified row storage cache is used for all data partitions in a partitioned table.

Once a table is enabled, `insert`, `select`, `update`, and `delete` statements may behave differently than in regular tables. The following table describes general behaviors affecting these statements:

Command	Row-Caching Off	Row-Caching On	Snapshot Isolation On
`insert`	Inserted in the page store.	New rows are inserted into the in-memory row storage..	New rows are inserted into the in-memory row storage.
`select`	Returns the row from wherever it resides.	ILM may cache rows in the in-memory row storage.	Depends on the `row_caching` subclauses.
`update`	Page store rows are updated in the page store, without row migration. Rows already in in-memory row storage are updated there.	ILM may migrate row to the in-memory row storage.	Row from the page store are migrated to, and updated in, the in-memory row storage. Rows already in the in-memory row storage are updated there.
`delete`	Page store rows are deleted from the page store, without row migration. Rows already in in-memory row storage are deleted from there.	Rows in the page store are deleted from wherever the row resides. Migrated rows are deleted from the in-memory row storage. Cached rows are discarded, and the page-store version of the row is deleted.	Rows from the page store are migrated to, and deleted from, the in-memory row storage. Rows already in in-memory row storage are deleted from there.

This example creates the `sales_detail` table, which uses datarow locking scheme and has row caching enabled:

```
create table sales_detail
(<column_names>)
lock datarows
with row_caching on
```

With row caching enabled database-wide in the `pubs2` database, if you create the `sales` table, it inherits the row-caching policy from the database, and there is no need to include this parameter in the `create table` command:

```
create table sales
(<column_names>)
lock datarows
```

3.12.1 Creating Tables the Use Row Storage Cache and Deferred Allocation

Use the row storage cache with deferred allocation to create fully memory-resident user tables.

For example, in a workflow queuing system that requires a table to store generated work orders, once each work order is processed and completed, the row for the work order is deleted from the table. This table may be a target of high-volume inserts, updates, and deletes. Over time, the table includes a lot of activity but at any single point of time, the table may not have very many rows, which makes it a good candidate for in-memory row storage.

In conjunction with deferred allocation, the table may often have no footprint on the disk for the data pages, thereby not incurring any I/O costs or page-concurrency issues (although index pages may still be written). This table is created using syntax similar to:

```
create table inmem_workflow_queue
(...)
lock datarows
with row_caching on, deferred_allocation
```

If sufficient memory is available, all rows go directly to the in-memory row storage and are processed in-memory. The `deferred_allocation` parameter ensures that no data pages are allocated to the table when it is created.

Tables with a large number of partitions may allocate a smaller number of data pages and extents when they are created. For example, deferred allocation can be an advantage in a multi-tenant application hosted by a table in the database, where different tenants are mapped to individual partitions. If the system suffers from low memory, the packing system is deployed, and data pages are allocated to the table only when rows are moved from the in-memory row storage to the page store.

Essentially, with a sufficient amount of memory available, a table that uses row storage cache with deferred allocation acts as a fully memory-resident queuing system that provides high performance with very little concurrency issues while accessing the data rows. In addition, you can deploy all other T-SQL features (for example, rules, defaults, privilege protection, and predicated privileges) on this table without any application changes.

See *Transact-SQL Users Guide* > *Databases and Tables* > *Deferred Table Creation* for information about deferred tables.

3.12.2 Altering Tables to Use Row Storage

Before configuring a table to use IMRS, enable its database for IMRS first, then use `alter table` to configure a table to use IMRS.

Deploying a row storage cache requires additional memory. The ideal configuration is to enable IMRS usage for critical "hot" tables that are heavily updated in the user's workload. Even when the IMRS is enabled database-wide, it only applies to newly created tables. Use `alter table` to enable the IMRS for datarows cache or MVCC functionality.

For example, if the `pubs2` database is already enabled for row storage, this command alters the sales table for row caching:

```
alter table sales
set row_caching on
```

See the *Reference Manuals: Commands*.

3.12.3 Creating Indexes on Tables with In-Memory Rows

There are a number of items to consider when you create indexes on in-memory enabled tables.

- B-tree indexes can point to rows that exist in the page store and to others that exist in the IMRS.
- There is no special syntax required to create indexes in IMRS-enabled tables.
- The following are supported when creating indexes in IMRS-enabled tables:
  - Global and local indexes on IMRS-enabled partitioned tables.
  - Index compression.
  - All existing index properties (for example, UNIQUE, ascending and descending indexes, `fill_factor`, `allow_dup_row`, `ignore_dup_key`).
- You can either build an index by loading data first into the IMRS page store, or you can build an index first, then insert rows.
- There are slight differences about where and how the data ends up when you build a new index:
  - Issuing `create clustered index` on a table with rows in the IMRS first moves rows to the page store, then builds the index.
  - The rows remain in the IMRS when you create a non-clustered indexes.
- Indexes on existing tables continue to be valid after the table is altered to use row storage. You need not drop and re-create indexes while migrating existing tables to use the IMRS, or if you disable row storage or row caching on tables.

For more information about creating indexes for IMRS-enabled tables, see the *Reference Manual: Commands > create index > Considerations for Creating Indexes on In-Memory Row Storage Tables*.

IMRS-enabled tables also support in-memory hash-cache B-tree indexes, which are used as performance accelerators for index scans. Hash indexes are created on unique indexes and are used as an access method by fully qualified scans on unique indexes. Hash indexes can be created on DRC-enabled and MVCC-enabled tables.

See *Performance and Tuning: Locking and Concurrency Control > Hash-Cache B-tree Indexes*.

3.13 Dropping Tables and Indexes from the In-Memory Row Storage

Dropping a table or index from an IMRS-enabled database is the same as dropping them from a regular database.

Use the `drop table` command to remove a table from the IMRS-enabled database. The syntax is:

```
drop table <table_name>
```

Issuing `drop table` in an IMRS-enabled database:

- Drops all pages for the table and deletes all rows from the IMRS belonging to this table, including older versions.
- Drops all indexes belonging to this table, and purges the in-memory structures of all rows belonging to this table used to navigate from the index to the row in the IMRS.

- Reclaims the memory used for rows in the IMRS for the table being dropped after the operation commits by the IMRS garbage collectory. You can track memory used by a dropped object in the monitoring tables until the memory is fully reclaimed.

Use the `drop index` command to drop an index from an IMRS-enabled database. The syntax is:

```
drop index <table_name>.<index_name>
```

Dropping an index does not result in rows moving from the IMRS to the page store.

There are no additional privileges or auditing requirements to drop an IMRS-enabled table or index other than the privileges required to drop an ordinary index or table.

See the appropriate SAP ASE reference manual for information about required permissions for running individual commands and system procedures.

3.14 Administering the IMRS

There are a number of IMRS administrative tasks that require special consideration.

3.14.1 Managing the Workload with sp_wlprofiler

Use the workload profiler to monitor and configure resources for the IMRS.

> **i Note**
>
> You must install the `sybdsamdb` database before using the workload profiler. Use the `installsybdsamdb` installation script (located in `$SYBASE/$SYBASE_ASE/scripts` on Unix; `%SYBASE%\%SYBASE_ASE%\scripts` on Windows) to install the `sybdsamdb` database.
>
> ```
> $SYBASE/$SYBASE_OCS/bin/isql < $SYBASE/$SYBASE_ASE/scripts/installsybdsamdb
> ```
>
> You must grant `select into` permissions on this database after installing the `sybdsamdb` database:
>
> ```
> sp_dboption 'sybdsamdb', 'select into', true
> ```

The workload profiler consists of a collection of tables and views located in the `sybdsamdb` database (all with the `wlp_` prefix), and a collection of procedures located in `sybsystemprocs`. These tables are installed when you install the workload profiler with the `installwlprofiler` script. However, the workload profiler is not installed by default. To install the workload profiler, use `isql` to run the `installwlprofiler` script, located in the `$SYBASE/$SYBASE_ASE/scripts` directory:

```
$SYBASE/$SYBASE_OCS/bin/isql < $SYBASE/$SYBASE_ASE/scripts/installwlprofiler
```

By default, runtime metrics gathered by the profiler are archived in the `sybdsamdb` database, which also contains the control tables. Make sure there is sufficient space in the metrics database to store the metrics collected during the run.

See the *Reference Manual: Tables* for a complete description of the workload profiler tables.

Following is a sample use-case scenario that manages a workload for the datarows cached feature using `sp_wlprofiler`:

1. Run `sp_wlprofiler monitor` to monitor the database (the `tpcc` database in this example). The recommended interval is 5 minutes; use shorter intervals if the workload spans a long period of time, you should take the monitor sample at shorter intervals. For example:

```
sp_wlprofiler 'monitor', 'tpcc', "USING interval='00:05:00'"
go
Spid 1691, Profiling Workload 2. The workload will be monitored for an
interval of '300' seconds after which the monitoring will 'end' ...
Warning: Some output column values in the result set may be incorrect.
Running 'UPDATE [TABLE] STATISTICS' may help correct them.
Archived 24 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_object' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_syslogs' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 50 rows to archive table 'sybdsamdb.dbo.WLP_monCachedObject' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 2 rows to archive table 'sybdsamdb.dbo.WLP_monDataCache' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 5 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceIO' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 10 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentIO' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 10 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentUsage'
with timestamp 'Nov 23 2016  2:44AM'.
Archived 24 rows to archive table 'sybdsamdb.dbo.WLP_monOpenObjectActivity'
with timestamp 'Nov 23 2016  2:44AM'.
Archived 24 rows to archive table
'sybdsamdb.dbo.WLP_monOpenPartitionActivity' with timestamp 'Nov 23 2016
2:44AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTableCompression' with
timestamp 'Nov 23 2016  2:44AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTempdbActivity' with
timestamp 'Nov 23 2016  2:44AM'.

Sample MDA metrics, Workload ID 2, counter 100
Archived 3555 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Nov 23 2016  2:45AM'.
Archived 30 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Nov 23 2016  2:45AM'.

Sample MDA metrics, Workload ID 2, counter 101
Archived 3579 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Nov 23 2016  2:46AM'.
Archived 30 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Nov 23 2016  2:46AM'.

Sample MDA metrics, Workload ID 2, counter 102
Archived 3589 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Nov 23 2016  2:47AM'.
Archived 30 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Nov 23 2016  2:47AM'.

Sample MDA metrics, Workload ID 2, counter 103
Archived 3594 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Nov 23 2016  2:48AM'.
Archived 30 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Nov 23 2016  2:48AM'.

Sample MDA metrics, Workload ID 2, counter 104
Archived 3597 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Nov 23 2016  2:49AM'.
```

```
Archived 30 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Nov 23 2016  2:49AM'.
Warning: Some output column values in the result set may be incorrect.
Running 'UPDATE [TABLE] STATISTICS' may help correct them.
Archived 24 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_object' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_syslogs' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 50 rows to archive table 'sybdsamdb.dbo.WLP_monCachedObject' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 2 rows to archive table 'sybdsamdb.dbo.WLP_monDataCache' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 5 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceIO' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 10 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentIO' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 10 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentUsage'
with timestamp 'Nov 23 2016  2:49AM'.
Archived 24 rows to archive table 'sybdsamdb.dbo.WLP_monOpenObjectActivity'
with timestamp 'Nov 23 2016  2:49AM'.
Archived 24 rows to archive table
'sybdsamdb.dbo.WLP_monOpenPartitionActivity' with timestamp 'Nov 23 2016
2:49AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTableCompression' with
timestamp 'Nov 23 2016  2:49AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTempdbActivity' with
timestamp 'Nov 23 2016  2:49AM'.
(return status = 0)
```

2. Monitor the workload using multiple iterations of the `monitor` parameter. All subsequent iterations of the `monitor` parameter use the same workload ID, interval, and sample time, and continue to collect metrics for the same monitoring session.

```
sp_wlprofiler 'monitor', 'tpcc'
```

3. After the workload has been monitored (which includes all the monitor samples), run `sp_wlprofiler finish` to mark the end of this workload session. This also performs the end processing, which includes workload planning based on the feature:

```
sp_wlprofiler 'finish'
Running plan for Workload Name='TPCC' affecting target db = 'tpcc' evaluating
feature(s) 'default'
Tables which qualify for Workload Name = 'TPCC' (ID = 2) affecting database =
'tpcc' evaluating feature 'default' are listed below:
 WLP_ID DBName Owner TableName  AvgRowLen NumRows   RowsInIMRS IMRSCache_MB
IMRSLog_MB
 ------ ------ ----- ---------- --------- --------- ---------- ------------
----------
      2 tpcc   dbo   district         106      2400       2400
2.43    1063.83
      2 tpcc   dbo   history           56  38855405    1942770
415.02     280.71
      2 tpcc   dbo   item              96    100000      83222
20.32       0.00
      2 tpcc   dbo   new_order         13   2744912    2645272
484.83     384.52
      2 tpcc   dbo   order_line        62 393463431   19673172
4208.49    3696.05
      2 tpcc   dbo   orders            29  31644052    1582202
290.18     275.84
      2 tpcc   dbo   stock            312  24000000    1200000
565.91    2417.10
      2 tpcc   dbo   warehouse        101       240        240
0.94     505.29
(8 rows affected)
```

```
Total IMRS Cache and IMRS log sizes needed for the workload:
WLP_ID DBName IMRSCacheSizeMB IMRSLogSizeMB
 ------ ------ --------------- -------------
      2 tpcc         12241.240     86233.400
(1 row affected)
(return status = 0)
```

4. Issue the `report` parameter to determine the qualified tables, IMRS cache size, and IMRSLOG size for the feature (tables that have a score above a specified cut-off are qualified for the feature).

```
sp_wlprofiler 'report'
Tables which qualify for Workload Name = 'TPCC' (ID = 2) affecting database =
'tpcc' evaluating feature 'default' are listed below:
 WLP_ID DBName Owner TableName  AvgRowLen NumRows   RowsInIMRS IMRSCache_MB
IMRSLog_MB
 ------ ------ ----- ---------- --------- --------- ---------- ------------
----------
      2 tpcc   dbo   district         106      2400       2400
2.43    1063.83
      2 tpcc   dbo   history           56  38855405    1942770
415.02     280.71
      2 tpcc   dbo   item              96    100000      83222
20.32       0.00
      2 tpcc   dbo   new_order         13   2744912    2645272
484.83     384.52
      2 tpcc   dbo   order_line        62 393463431   19673172
4208.49    3696.05
      2 tpcc   dbo   orders            29  31644052    1582202
290.18     275.84
      2 tpcc   dbo   stock            312  24000000    1200000
565.91    2417.10
      2 tpcc   dbo   warehouse        101       240        240
0.94     505.29
(8 rows affected)

Total IMRS Cache and IMRS log sizes needed for the workload:
WLP_ID DBName IMRSCacheSizeMB IMRSLogSizeMB
 ------ ------ --------------- -------------
      2 tpcc         12241.240     86233.400
(1 row affected)
(return status = 0)
```

5. Issue the `report config` parameter to determine the DDL's required to configure the IMRS cache, create the on-disk row storage device, and alter the database and qualifying tables to enable IMRS:

```
sp_wlprofiler 'report config'
List of DDLs for Workload Name = TPCC (ID = 2) affecting database = tpcc
evaluating feature 'default'.
To apply these DDLs automatically and reconfigure the ASE server, use the
sp_wlprofiler 'reconfigure' sub-command.
------------------- BEGIN -------------------------
use master
go
-- Create In-memory Row Storage Cache.
-- Before executing this, the max memory needs to be increased by 12241MB
--
sp_cacheconfig 'tpcc_imrs_cache', '12241M', row_storage
go
-- Create the IMRSLOG device.
-- The physical location of the device needs to be updated
-- before running the command.
--
disk init name = 'tpcc_imrs_log_dev'
        , physname = '<full_path>/tpcc_imrs_log_dev'
        , size = '86233M'
        , type = imrslog
go
```

```
ALTER DATABASE tpcc IMRSLOG ON tpcc_imrs_log_dev = '86233M'
go
ALTER DATABASE tpcc ROW STORAGE ON tpcc_imrs_cache
go
use tpcc
go
ALTER TABLE dbo.history SET ROW_CACHING ON
go
ALTER TABLE dbo.order_line SET ROW_CACHING ON
go
ALTER TABLE dbo.warehouse SET ROW_CACHING ON
go
ALTER TABLE dbo.district SET ROW_CACHING ON
go
ALTER TABLE dbo.stock SET ROW_CACHING ON
go
ALTER TABLE dbo.orders SET ROW_CACHING ON
go
ALTER TABLE dbo.new_order SET ROW_CACHING ON
go
ALTER TABLE dbo.item SET ROW_CACHING ON
go

------------------- END ------------------------
(return status = 0)
```

3.14.2 Determining the imrslog Threshold

Transactions generate `sysimrslog` entries as they insert or modify rows in the IMRS.

The `sysimrslog` is responsible for persisting changes to the IMRS, and for maintaining a consistent state for the IMRS across server crashes, dumps, and loads.

> **i** Note
>
> Truncate the log to prevent overflow, since `sysimrslog` continues to grow because records are appended as the IMRS performs transactions Truncation is possible because only the transaction that inserted the oldest row in the IMRS to the last IMRS transaction is required to perform recovery, not the entire `sysimrslog`.

The IMRS-related information for transactions is stored in a descriptor holding IMRS-related metadata for transactions committing to `sysimrslogs`, and the list of all the rows modified (or inserted) by transactions is included in this descriptor as a list of IMRS row versions. When a transaction modifies a row in the IMRS, the garbage collector collects the older IMRS row versions. Once all the versions belonging to this descriptor are collected, the descriptor itself is collected.

DBTABLE maintains a queue of the descriptor structures. It is populated at commit-time and represents a chronological ordering of transactions, with the oldest inserted transaction occupying the top of the queue (that is, the oldest element in queue). In an ideal `sysimrslogs` truncation, the oldest inserted transaction is moved forward to the next descriptor when the garbage collector culls the descriptor at the head of the queue, updating the marker.

Occasionally, a row inserted in the IMRS is not updated, so the oldest inserted transaction cannot move forward, or the insert rate is greater than the rate at which older rows are modified. When this happens, the server must intervene to prevent IMRS cache and `sysimrslogs` from clogging.

The IMRS_PACK threads pack rows from the IMRS if cache utilization goes beyond 70 percent (the value of `cache_util_thresh`), and uses aggressive pack if the utilization goes beyond halfway between `cache_util_thresh` and 100 percent. At 70 percent utilization, the halfway point is 85%. This packing runs automatically, and occurs according to the ILM rules. That is, it packs the rows that are considered cold (not frequently used) regardless of when they where first inserted or updated in the IMRS.

The threshold for `sysimrslogs` is set to 50 percent of the space used from `sysimrslogs`. When this threshold is crossed, the `sp_thresholdaction` system procedure packs rows out of the IMRS, so the percentage of available space in `sysimrslogs` after `sp_thresholdaction` runs is:

```
50% + @@imrslogpackpct
```

This packing occurs with regard to the `imrslog`. That is, it first packs all the rows associated with the oldest inserted transaction, then the next descriptor, and so on until the desired free space limit is reached.

If the rows associated with the oldest inserted transaction are frequently used, the IMRS_PACK will not pack or move them, so that space in `sysimrslogs` may not become free. It will, however, free up space in the IMRS cache The pack operation requires writing the data pages and may be slower if the pages are not already in cache.

To guarantee the freeing of space, use `sp_imrslog_thresholdaction` instead, as it packs rows according to `sysimrslogs`. Since `sp_imrslog_thresholdaction` packs rows serially (starting from the oldest inserted transaction), it may pack some rows that are not cold according to the ILM rules. This should not affect the system because hot rows are frequently touched and are brought back to IMRS.

The pack operation is a cross-logged transaction, so logs are written to `syslogs` and `sysimrslogs` during pack operations. Consequently, issuing `sp_imrslog_thresholdaction` invokes the `syslogs` threshold, but as `syslogs` fills, periodic checkpoints truncate the log.

3.14.2.1 How IMRS Packs Rows

When it is created, an IMRS-enabled database uses the value for `<@@imrslogthpct>` to establish a threshold for `sysimrslogs` for the amount of free space, and it configures a value for `sp_imrslog_thresholdaction` that corresponds to this value.

IMRS-enabled database uses a last-chance threshold of 5 percent of free space to determine when it invokes `sp_imrslog_thresholdaction`. This percentage is not configurable.

While the system runs and the amount of free space continues to shrink, the threshold manager invokes `sp_imrslog_thresholdaction` when the amount of `sysimrslogs` free space reaches the default 50 percent. `sp_imrslog_thresholdaction` determines the current value for `free_space` (which should be approximately 50 percent because the threshold was fired) and computes the amount of free space required when `sp_imrslog_thresholdaction` completes using this formula:

```
free_space in sysimrslog = (@@imrslogthpct + (@@imrslogpackpct% of
@@imrslogthpct))
```

For example, using the default settings, `sp_imrslog_thresholdaction` packs rows until the percentage of `sysimrslogs` pages are free or inactive (and can be truncated by the next `dump transaction` command):

```
50% + (10% of 50%) => 50% + 5% => 55%
```

Once `sp_imrslog_thresholdaction` determines the threshold for space_usage, it repeats the operation by packing a single allocation unit of `sysimrslogs`, which is the minimum amount of space that is required for a `sysimrslogs` truncation.

The server cannot run multiple `sp_imrslog_thresholdaction` or `sp_thresholdaction` system procedures for the same threshold simultaneously. Once one of these procedures starts, it runs until it has packed rows up to the value of `<@@imrslogpackpct %>`, the hysteresis. Space usage becomes greater than the threshold mark when the procedure finishes, and it does not run again until the next time the amount of free space falls below the threshold.

Packing an allocation unit results in one of the following:

- Successful pack with at least one row packed – because all rows are successfully packed in a single allocation unit, the oldest inserted transaction moves forward and the server moves to the next iteration, packing the next oldest allocation unit, and repeating this sequence until it reaches the desired amount of free space.
- Nothing to pack in this allocation unit because everything is queued to the garbage collector – if there is nothing to pack in the current allocation unit, `sp_imrslog_thresholdaction` retries to repack it 100 times (non-configurable) with a one-second sleep between attempts. If the result for all consecutive 100 retries is `0 rows packed`, `sp_imrslog_thresholdaction` stops because it determines that the garbage collector cannot free the IMRS row versions and the descriptor holding IMRS-related metadata for transactions committing to `sysimrslogs` in the time allotted.
- An error occurs during pack (for example, a lock conflict) – similar to the second situation, `sp_imrslog_thresholdaction` retries 100 times with a one-second sleep between attempts to pack the allocation unit. If the result for all consecutive 100 retries is `0 rows packed`, `sp_imrslog_thresholdaction` stops because it determines there is something wrong with the transaction that is causing the locked rows to not be released after 100 seconds. This prevents the threshold procedure from running indefinitely.

After the pack operation, `sp_imrslog_thresholdaction` calls `sp_thresholdaction`, configure it to call `dump transaction` to truncate all the allocation units freed during the pack operations, and to reclaim `imrslog` space. Doing so removes rows from the IMRS cache and pushes them to the page store, which moves the oldest inserted transaction forward. However, the log truncation occurs only when the PTNFIRST moves from a `dump transaction`.

3.14.2.2 Tuning Your imrslog Threshold

Parameters allow you to tune `imrslog` thresholds.

> **i** Note
>
> The minimum size of `imrslog` should be twice the size of IMRS-cache. A server can generally sustain an `imrslog` that is five times the size of the IMRS-cache. THis allows the server to invoke the IMRS_PACK threads before running `sp_imrslog_thresholdaction`.

Parameter or System Procedures	Desription
<@@imrslogthpct>	Determines the percentage of free space with which imrslog threshold is installed. This percentage is non-configurable. Default value: 50%.
<@@imrslogpackpct >	Determines the threshold level above which sp_imrslog_thresholdaction addresses the free_space in the imrslog segment. Default value: 10%. To modify, use this dbcc command: `dbcc imrs('set','imrslog', <database_name>, 'packpct', <new_%>)`
Trace flag 7439	Installs a threshold procedure for sysimrslogs every iteration of this formula, starting at 50 percent of the imrslog size: `@@imrslogpackpct % (defaults 10%)` Default values: 50%, 40%, 30%, 20% and 10% of sysimrslog free space.
sp_imrslog_thresholdaction	A user-configurable system procedure that is bound to the imrslog threshold. This not configurable by the user, and is predefined to pack a designated number of rows.
sp_thresholdaction	System procedure called at the end of sp_imrslog_thresholdaction. This procedure is user configurable. Configure sp_thresholdaction to call a dump transaction on the target database to truncate the sysimrslog pages.

Use the select command to view the parameter values. For example:

```
select @@imrslogthpct
 ---
  50
```

3.14.3 Interactions with Other Database Options

There are a number of sp_dboption parameters that require special attention in an IMRS-enabled database.

- abort tran on log full – supported for IMRS-enabled databases. When enabled, the transaction is aborted if there is insufficient space in sysimrslogs. Currently executing transactions are aborted when the last-chance threshold (LCT) in sysimrslogs is crossed. When this option is disabled, currently

executing transactions are suspended, but are awakened when free space is made available in `sysimrslogs`. Transactions affecting IMRS-enabled tables may not see an out-of-`imrslog` situation until late in the transaction because logging to the `sysimrslogs` is done as part of the commit-time operation. Regardless, the transaction is rolled back.

- `allow incremental dumps` – incremental dumps of an IMRS-enabled database are not supported. This option is disabled if you are altering a database to be IMRS-enabled.
- `deferred table allocation` – not supported for IMRS-enabled databases. If this option is enabled in an existing database that is subsequently altered for IMRS usage, the `deferred table alloction` option is not inherited by any newly created tables in the database.
- `delayed commit` – supported for IMRS-enabled databases, but is restricted to flushing `syslogs` pages. When this option is enabled, the changed data is persisted to `sysimrslogs` synchronously when `delayed commit` is enabled at commit-time. Consequently, you may not realize the full benefits of `delayed commit` when a transaction spans both sides of the storage regions.
- `enforce dump tran sequence` – supported for IMRS-enabled databases. However, because `enforce dump tran sequence` affects transaction logging on changes done to the page store, there is no benefit for enabling this option on logging operations done to `sysimrslogs`.
- `erase residual data` – supported for IMRS-enabled databases. However, in-memory data in the IMRS is not cleansed following a `delete`, `truncate`, or other event that removes IMRS data rows from a table. Since `erase residual data` is intended for data that is persisted on pages, enabling it has no affect on cleansing in-memory data.
- `full logging for {all | alter table | reorg rebuild | select into}` – supported for IMRS-enabled databases. However, because `full logging. . .` applies to logging to the page store, enabling it has no affect for changes applied to `sysimrslogs`. If any utility requires data rows to be packed from the IMRS to the page store, the page-store side of changes of the utility are fully logged in `syslogs`, where `full logging. . .` is enabled.
- `no chkpt on recovery` – supported for IMRS-enabled databases. Checkpoint records are not written to the log at the end of recovery when `no chkpt on recovery` is enabled, allowing additional `load transaction` commands to be executed.
- `read only` – supported for IMRS-enabled databases. Any updates to the IMRS or `sysimrslogs` are prevented when `read only` is enabled.
- `trunc log on chkpt` – supported for IMRS-enabled databases. Space in `sysimrslogs` is truncated after a checkpoint when `trunc log on chkpt` is enabled, subject to recoverability requirements of the IMRS.
- `allow wide dol rows` – supported for IMRS-enabled databases. Enable this option on servers that use 16K page sizes to insert wide rows, or update existing rows to create wide rows, on data-only-locked tables in the IMRS.
- `async log service` – supported for IMRS-enabled databases but inapplicable to the IMRS because `async log service` only helps with flushing the user log cache for `syslogs`.
- `ddl in tran` – supported for IMRS-enabled databases, but does not apply in an IMRS environment since all DDL changes and logging is performed in system tables and `syslogs`.
- `no free space acctg` – supported for IMRS-enabled databases, but does not apply in an IMRS environment since this option suppresses free-space accounting and threshold procedures for data segments, and the logsegment and `imrslogsegment` are not affected.
- `scratch database` – supported for IMRS-enabled databases, but does not apply in an IMRS environment. Although you can use IMRS-enabled databases as scratch databases for loading archives,

there is no value in doing so because a scratch database is used only as a shell database, and only used to instantiate `sysaltusages`.

- `select into` – supported for IMRS-enabled databases, but applies only when inserting rows with minimal logging directly into the page store. There is no need for minimal logging when you issue `select into` to create an IMRS-enabled table.
- These database options are fully supported for IMRS-enabled databases:
  - `auto identity`
  - `dbo use only`
  - `deallocate first text page`
  - `identity in nonunique index`
  - `single user`
  - `unique auto_identity index`

3.14.4 Using BCP in an IMRS Database

Use `bcp` to copy data out of an IMRS-enabled table just as you would copy data out of a regular table.

Consider the following when you use `bcp` to load data in an IMRS-enabled table:

- Fast `bcp` – use fast `bcp` when the table has no indexes or triggers (and few other restrictions: see the *Utility Guide*), and the `select into/bulkcopy` database option is enabled. All the data rows are inserted in bulk mode into newly allocated data pages. To retain the behavior of fast data loads into a table without indexes, issue `bcp` to an IMRS-enabled table with conditions that cause a regular table to use fast `bcp`. Typically, this means inserting directly in bulk mode to newly allocated data pages, bypassing the IMRS.
- Fully logged Fast `bcp` – when the table has one or more non-clustered indexes, and other conditions that disallow using a fast `bcp` are disabled (for example, there are no triggers), the data is loaded directly in bulk mode to the page store pages. The server rebuilds the nonclustered indexes (in parallel when possible). Inserts go directly to the page store for IMRS-enabled tables that satisfy these conditions, bypassing the IMRS.
- Slow `bcp` – when the table has conditions that prevents using fast `bcp` (for example, the table includes indexes, triggers, or `select into/bulkcopy` is disabled), `bcp` inserts data one row at a time, using regular insert handling. For an IMRS-enabled table, these rows are, however, inserted directly to the page store.
- After loading data using `bcp`, if data rows are frequently accessed or updated, they are automatically brought back into the IMRS.

3.14.5 Using LOB Columns in the IMRS

IMRS-enabled tables can also include off-row or in-row, large object (LOB) columns.

Consider the following when you are including LOB columns in IMRS-enabled tables:

- In-row LOB length criteria and limits relative to the server's page size continue to apply even though an in-memory row could, theoretically, be larger than the logical page size.
- In-memory rows with off-row LOB columns have the same row format as an on-disk row, with an embedded text pointer to the first text page (FTP) of the off-row disk-based LOB page chain.

- Updates of in-row LOB columns to inserted or migrated rows are performed by producing new, in-memory row versions of the expanded in-row LOB column. The new row version stores the column value in memory as long as the updates to the in-row LOB columns allow it to be retained in-row.
- When an update of a LOB column, or updates of other variable-length columns, causes an in-row LOB value to move off-row, the new row-version created contains the text pointer to the FTP of the off-row LOB page chain. The movement of the LOB value from in-row to off-row is a fully logged operation in the page store, and is captured in `syslogs`. However, the changes to the in-row component of the LOB columns is captured in `sysimrslogs` as part of the `commit`.
  Conversely, shrinking an update of an in-row LOB column's value that is currently off-row migrates that value back in-row. This results in a page deallocation of the off-row LOB page chains, logged in `syslogs`, and the new in-memory row version is created with the updated smaller in-row LOB value.
- LOB updates combined with row migration can produce situations that require special consideration:
  - Inserted row contains only in-row LOB columns – no further work needs to be performed for the delete.
  - Migrated rows (even if there are only all in-row LOB columns) – the `delete` operation of the counterpart row in the page store steps through all off-row LOB columns and deallocates their page chains. Because the row in the IMRS may no longer have off-row LOB columns—but the original may have had several—the server must optimize to carry forward the state of each LOB column that was off-row when it was migrated to the IMRS, driving the page deallocation of those LOB columns which were off-row before migration.
  - Off-row LOB columns with off-row LOB values – The in-memory row may have some off-row LOB columns, and some of these may also have off-row LOB values in the counterpart page-store row. This can occur when a row with off-row LOB values is migrated to the IMRS, followed by several updates that bring the off-row values in-row. Subsequent updates cause the server to migrate these in-row values off-row. Deleting this row from the IMRS requires deallocating the off-row page chains of the latest version of the row along with deallocating the off-row page chains of the page-store counterpart version of the row.
  - Frequent updates to in-memory rows contain uncommitted versions with off-row LOB chains for some, but not all, columns. In this situation, the LOB values may move from off-row to in-row across the updated versions, and the row is deleted within the same transaction. Every intermediate update to an off-row LOB column is performed as a `delete` then an `insert` operation, so the page-deallocation of off-row LOB columns in older versions is already preformed in-line with the statement. The memory reclamation of these intermediate versions is performed by a background garbage collector thread, which hands off the FTP handle for embedded off-row columns to another LOB page deallocation background thread.

3.14.6 Using Triggers in the IMRS

IMRS-enabled tables support all forms of triggers, however there are differences in the way the IMRS system processes triggered inserts and deletes.

DML operations on rows in the IMRS do not generate in-line logging, and write log records only at commit-time to `sysimrslogs`. However, triggers are fired at the end of a statement, at a time when log records have not yet been generated.

Although the mechanism for firing the triggers does not change, the mechanism for generating the rows in the inserted and deleted tables referenced in a trigger body (called pseudo tables) changes slightly in an IMRS system.

In regular tables, the rows for these pseudo tables are synthesized by the equivalent log records from `syslogs`. For an IMRS-enabled table, the same data is synthesized using the in-memory row versions that are maintained in the IMRS. Because runtime logging is not performed for DMLs on IMRS-enabled tables, and the server need not access or scan `syslogs` to generate the rows for the inserted and deleted tables, trigger performance is expected to be slightly improved when triggers are fired on an IMRS-enabled tables.

For a DRC-enabled table, a single DML statement (for example, `update` or `delete`) may affect rows both in the page store and in the IMRS. For the same statement, updates or deletes may be done in the page store without migrating the rows to the IMRS, and other changes may be performed to rows that already reside in the IMRS. Even without completely ILM-aware data row caching strategies, when row caching is disabled, a `merge` statement may inset new rows in the page store, and the `update` part of the `merge` may qualify rows that are already residing in the IMRS. This results in some changes performed with logging in `syslogs`, and others performed with in-memory versioning. In this situation, a trigger that is fired on the affected table can synthesize the data for the inserted and deleted tables by accessing the changes from `syslogs` where applicable, and from the in-memory row versions for the updates, inserts, or deletes performed in the IMRS.

The IMRS system supports instead-of-triggers defined on views created on datarow cache-enabled tables. However, LOB columns can be referenced in these pseudo-tables inside the body of an instead-of-trigger defined on views. DMLs on views defined with instead-of-triggers fire the corresponding trigger, and the affected rows are generated from the plan-private worktable-based cache, including for LOB columns.

3.14.7 Committing Transactions in an IMRS

IMRS log records are generated and flushed to `sysimrslogs` at commit-time, which allows for aggregated logging and reflect the final state of a row if it was updated multiple times in a single transaction, and allows for in-memory rollback without requiring additional logging. That is, `sysimrslogs` captures only redo log records.

All log records that affect in-memory rows must be generated as part of commit processing. This involves "visiting" each of the affected rows and fabricating its log record to reflect the changes caused by the transaction. If several hundreds, or thousands, of rows are affected in a transaction, the cost of generating the log records may appear to be significant.

Some scalability and performance concerns may arise due to in-memory row structure traversal at commit-time. Multiple transactions may be committing simultaneously on different cores, causing some cache-line hits on high-end multicore machines. Although the affected row is readily accessible in non-IMRS databases, in IMRS-enabled databases, the server must traverse some transaction-local memory chains to locate the affected rows at commit time, and traversing long in-memory row version chains during during performance-critical commit-processing may cause scalability issues.

Commit-time logging is performed concurrently across different transactions. These transactions may be committing simultaneously, allowing different transactions to write a section of the `sysimrslogs` page chains, reducing the overhead of synthesizing log records for all DML operations affecting in-memory rows during commit processing.

3.14.7.1 Rolling Back Transactions in an IMRS

IMRS-enabled databases support all forms of `rollback`, and all runtime rollback is performed using the in-memory row versions produced for DMLs.

Rolling back a transaction that includes changes to rows in the page store and in the IMRS is performed in two steps: the changes affecting rows in the page store are rolled back using inline log records in `syslogs`, then the changes affecting the rows in the IMRS are rolled back using in-memory row versions.

The memory for intermediate row versions for `commit` and `rollback` processing is not immediately released once the transaction completes, but is managed by the garbage collector thread, so runtime transaction throughput is unaffected.

3.14.8 Using Pack Operations to Move IMRS Rows to the Page Store

Pack is a data movement operation that is performed by pack threads to move cold, infrequently used or accessed data rows from the IMRS to the page store.

> **i** Note
>
> This section discusses the pack threads operations for a datarow column- or IMRS-MVCC-enabled tables; they do not apply to on-disk MVCC tables.

Moving cold data out of the IMRS frees memory occupied by this data so that it is available for newer data. At run-time, the IMRS tracks information to detect access patterns to hot pages or hot rows. The IMRS uses these access patterns to improve performance. When hot data becomes less frequently used in the IMRS, it is packed back to the page store.

You can pack rows as a background process or force them by using the `sp_imrs ... pack_rows` system procedure, specifying the number of rows to pack based on the number of allocation or as a percentage of the total rows in the IMRS.

Regardless of the method you use, the pack data movement operation is fully logged and recoverable. While the pack operation is executing, concurrent transactional activity is allowed in the database, except on the rows being packed. Memory previously used by cold data rows that have been packed is reclaimed by the IMRS garbage collection thread. The server uses this memory to process new, more active data.

Configuring the Background Pack Operation

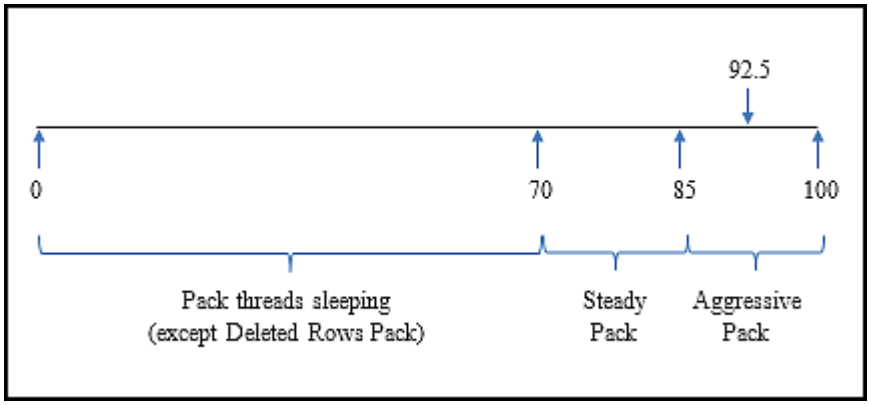
Information lifecycle management (ILM) and the background pack subsystems work cooperatively to maintain the IMRS cache-utilization value set by `imrs cache utilization`. Based on this value, the pack subsystem determines whether to use steady pack mode or aggressive pack mode to maintain cache utilization. For example, at low cache-utilization levels, the ILM may allow more rows to be brought into the IMRS, but at high cache-utilization levels, the pack threads may aggressively move data out of the IMRS to the page store.

The background pack operation runs automatically on IMRS-enabled tables. You can configure how aggressively these pack threads work to clear the cold data by changing the value of the `imrs cache utilization` configuration parameter.

When the current cache utilization crosses the configured value for `imrs cache utilization`, the pack threads wake up to perform a pack operation and select one of these pack modes:

- Steady pack mode – (most commonly used mode) the pack operation attempts to pack rows by applying ILM heuristics, which are based on partition-level counters and timestamps, and indicate whether a row is likely to be cold. The pack operation works in a steady pack mode when IMRS utilization goes beyond the value set by the `imrs cache utilization` configuration parameter. Only cold rows are packed in the steady pack mode. The pack operation continues in this mode until it reaches the threshold for aggressive pack mode.
- Aggressive pack mode – the pack operation works in an aggressive mode when the percentage utilization of IMRS cache is greater than the midway point between `imrs cache utilization` and 100. This mode is useful to quickly free space in the IMRS cache when memory utilization is high. In this mode, the pack operation packs rows without applying ILM heuristics to determine whether a row is hot or cold.

The pack operation chooses which mode to maintain cache utilization based on the value of `imrs cache utilization`. This example sets the `imrs cache utilization` to 70:



See the *Reference Manual: Configuration Parameters* for more information about `imrs cache utilization`.

The pack operation attempts to avoid out-of-memory situations by packing in-memory rows to the page store to free memory in the IMRS, so that application memory outages (for example, transaction or statement rollbacks) do not occur from incorrect ILM decisions.

The transactions continue to operate using the page store, which may result in slightly reduced performance, but do not result in loss of business continuity. Transactions are rolled back and the server issues an out-of-memory error when the pack operation cannot move rows to the page store, or there is a fragmentation and memory cannot be allocated. While in steady pack mode, ILM determines whether to pack a row (how "hot" it is). The pack operation packs only cold rows while it is in a steady pack mode.

Additional background pack operations:

- Early pack of deleted rows – pack operations run in this mode even when the percentage utilization of the IMRS cache is lower than the value of `imrs cache utilization` (70 in the example above) and is are a large number of deleted rows in at least one of the IMRS-enabled partitions in a database.
- Pack for RIDMAP page memory reclamation – RIDMAP pages can require large, contiguous memory chunks. Pack operations run in this mode when these contiguous pieces of memory have a low availability.

In this mode, a pack thread selects a RIDMAP page and packs all rows on it so that the page can be freed for reuse.

The pack operation determines which rows to pack at the data partition level, moving more rows from cold partitions. The `IMRS Pack` thread cycles through the IMRS-enabled tables and partitions that have rows in the IMRS. Each candidate partition's rows are evaluated for removal to the page store according to these metrics and heuristics:

- How frequently and how recently the rows are accessed – cold rows are packed to the page store. The pack operation uses row-usage metrics on a per-partition basis to select which rows from a table to select and pack to the page store, allowing row usage trends to help determine the rows being removed from the IMRS.
- Total memory consumption in the IMRS – pack operations are done more aggressively if memory usage thresholds are exceeded, sometimes overriding other heuristics about how hot the rows are to reclaim memory from more rows.
- Memory usage footprint for individual tables – the server may not pack rows from small tables as often as rows from larger tables, which occupy more memory in the cache.

Multiple IMRS pack threads can operate in a database. Use the `number of imrs pack tasks per db` configuration parameter to determine the number of threads assigned to a database. See the *Reference Manual: Configuration Parameters*.

Changes to the IMRS are logged in `sysimrslogs`. An action called `sysimrslog` truncation provides log space for newer activity by periodically reclaiming the space consumed by older, committed transactions The pack operation facilitates moving the truncation point in `sysimrslogs` since it cannot be truncated beyond the transaction containing the oldest insert row log record. The server can move the truncation point forward when these rows being held are relocated from their position in the log.

Manually Packing Rows

If the background pack system is ever insufficient and you need to immediately move rows to the IMRS, the `sp_imrs … pack_rows` parameter allows you to force the packing of rows to the IMRS according to the number of rows or the percentage of the rows in-memory.

3.14.9 Archive Management for ODRS

Database archives created using the `dump database` and `dump transaction` commands capture the contents of the on-disk row storage (ODRS) as part of the archive.

Reloading the archives using `load database` or `load transaction` reinstantiates the IMRS from the contents in the archive followed by recovery, as required, for the IMRS. However, the contents of the indexes are fully logged, and are recovered through the log records in syslogs.

Because the ODRS is part of the database physical storage region, archive management is provided with commands like `dump database`, `dump transaction`, `load database`, `load transaction`, and any extensions to the Backup Server methods like compressed dumps, and operations from the IBM Tivoli Storage manager.

Once recovery is complete, the index rows point to the new location of the data rows, which are resident in the IMRS. In-memory translation tables provide this transparent access to convert the RID for IMRS-resident rows to their in-memory addresses.

The load process does not configure the row storage cache: the cache must be created prior to the load operation. The IMRS expects that the target database was created to use an existing row storage cache of the appropriate size, which is at least as large as the IMRS cache size in the source database. The archive header contains information about the cache configuration on the source server, which is used to validate that the cache on the receiving server is adequately sized. The load process may still fail to restore the contents of the ODRS captured in the archive to the target row storage cache. The load operation does not attempt to reclaim memory from the cache by scavenging older rows and writing them to the page store.

3.14.10 Dumping and Loading into an IMRS-Enabled Database

IMRS-enabled databases support `dump` and `load database` and `dump` and `load transaction`, except for cumulative dump and loads.

Dumping an IMRS-enabled database dumps the contents of the database plus the contents of `sysimrslogs`. Successive `dump transaction` commands dump the active portion of `syslogs`, from the oldest open transaction to the last page of `syslogs` at the time the dump starts, plus the portion of `sysimrslogs` from the last committed transaction copied to a previous dump, or from the first page of `sysimrslogs` if the previous dump did not have any.

The archive captures the high-watermark size of the IMRS cache in its metadata. Based on this information, the `load` process validates the cache size required against the IMRS cache size provided in the target database. The `load` command fails if the target database provides an inadequate IMRS cache size.

You can reduce the IMRS cache size at runtime as long as the smaller size is above the current cache utilization. Consequently, you can perform a series of cache size increases and decreases before a `dump transaction` operation. The target database requires an IMRS cache large enough to hold the high watermark of the cache sizes between the `dump transaction` and its previous archive (which may be a `dump database` command).

The contents of the IMRS are recovered from `sysimrslogs`, but the page store contents are recovered from `syslogs`. Both stores are recovered concurrently.

When loading a database dump from an IMRS-enabled database, the target database must also be IMRS-enabled, its `sysimrslogs` must be large enough to hold the contents from the archive, and the IMRS cache must be at least the size of the HWM of the source database. You can, however, load an archive from a non-IMRS-enabled database into an IMRS-enabled database, in which case, there are no requirements for size of the `imrslog` or the size of the IMRS cache.

Alternatively, you can alter an existing disk-resident database to provide the `sysimrslogs` system table and IMRS cache, making the database an IMRS-enabled database. The next dump transaction contains the footprint of the ODRS in the archive.

Interactions with `dump` and `load` commands on IMRS-enabled tables with other commands:

- Using the `enforce dump tran sequence` database option does not impact IMRS-enabled databases. If this option is enabled, commands that perform minimal logging are disallowed, so you can still perform

`dump transaction` operations. All commands affecting rows in the IMRS are fully logged, including `select into`, and use commit-time logging, so it makes no difference if they are run in an IMRS-enabled database.

- Database options like `full logging for {all | alter table | select into | reorg rebuild}` control how the logging is performed in the page store (`syslogs`). Regardless of whether these options are set, the logging component for these options in `sysimrslogs` is unaffected and do not compromise a subsequent `dump transaction` operation.
- `load database with listonly=create_sql` reports the `sp_cacheconfig` and `create` or `alter database` commands needed to re-create the IMRS and enable the database to use the row storage cache.
- `load database with headeronly` reports the IMRS cache usage captured in the dump archive.
- Recovery of `create index` during a `load transaction` operation is restricted – for page store-enabled databases, `create index` changes are recovered from the transaction log as part of the `load transaction` operation. However for IMRS-enabled database, a index recovery is deferred until the `online database` phase completes, at which time the index is completely rebuilt using the fully recovered contents of the IMRS and the page store.
- Cross-platform dumping and loading of IMRS-enabled database and transaction log archives is not supported.
- You can use `load transaction` with the `until_time` parameter to load an IMRS-enabled database. However, the `until_time` parameter breaks the load sequence, and it cannot be followed by more `load transaction` commands. To restart the load transaction sequence, issue an `online database` or `load database` command.
- For a load and load-recovery to succeed, make sure the database or transaction log dumps from your IMRS-enabled database load into an IMRS-enabled database that has a large enough row storage cache. You cannot recover the contents of an IMRS-enabled database from the database or transaction log dumps directly into a disk-resident database.

3.14.11 Shrinking of IMRS Databases

Shrinking an IMRS database is not supported.

Shrinking all `imrslog` devices while the IMRS is enabled and assigned to the database is restricted. To shrink a database off the `imrslog` devices, deactivate the row storage for the database using `alter database row storage off`. You cannot shrink the on-disk row storage used for `sysimrslogs` while the IMRS is allocated to the database, even after you have moved data off the IMRS to the page store.

As an alternative to shrinking an IMRS database, drop all the IMRS devices instead. See Dropping IMRS Devices [page 89] for details."

3.14.12 Dropping IMRS Devices

You can drop all the IMRS devices.

Procedure

1. Unbind the IMRS cache from the database. This example unbinds the `pubs2` database:

   ```
   alter database pubs2 row storage off
   ```

2. Remove all database fragments from `imrslog` devices. For example:

   ```
   alter database pubs2 imrslog off
   ```

3. Use the `sp_dropdevice` system procedure to drop database devices allocated to `sysimrslogs`. For example:

   ```
   sp_dropdevice pubs2_odrs_cache
   ```

3.14.13 Using Compression with IMRS-Enabled Databases

IMRS-enabled databases can provide performance benefits while accessing compressed data rows. Once row caching is enabled, frequent access or updates to "hot" rows stored in an uncompressed form in the IMRS can deliver significant performance gains over the same operations on compressed rows in the page store.

You can configure IMRS-enabled databases for data, index, or LOB compression. Data row or page compression applies to data in the page store while the data in the IMRS remains uncompressed. If so configured, off-row LOB data continues to be compressed. Any data movement from the IMRS to the page store (for example, through the pack utility or the `alter table row_storage off` command), results in compressed data if the table is defined for data or LOB compression. IMRS-enabled tables are completely unaffected by index compression.

3.14.14 Interactions Between IMRS-Enabled Databases and Other Server Options

There are a number of items to consider when you are using IMRS-enabled databases with other server options.

- Security
  - Encryption
    - Full database encryption – data remains unencrypted when individual data rows are stored in the IMRS. When older data is packed from the IMRS to the page store, the data remains unencrypted in the data pages while in the buffer cache. These data pages are encrypted only when they are written to disk.

    - Column-level encryption – fully supported for data rows in the IMRS. Since column-level encryption and decryption is supported entirely in the query processing layer, when accessing individual columns' data from IMRS-resident data rows, the existing privilege checking is enforced to decrypt column values for privileged users, or to return unencrypted values or `decrypt_default` values for specified columns.
    - Data in `sysimrslogs` is not encrypted. However, log records in `syslogs` are encrypted.
  - Secure store – fully supported in IMRS-enabled databases.
  - Row-level access control – fully supported in IMRS-enabled databases.
  - Predicated privileges – fully supported in IMRS-enabled databases.
- Component integration process (CIS) – Not supported. You cannot assign row storage caches to proxy tables or proxy databases.
- High availability – Not supported.
- Distributed Transaction Management (DTM) – restricted use in IMRS-enabled databases. A distributed transaction spanning a regular table and an IMRS-enabled table can span multiple IMRS-enabled databases as long as it does not perform DMLs on IMRS-enabled tables, even if the DML affects only rows in the page store.
- Suspect granularity – Not supported.
- Cluster Edition – Not supported.
- Replication Server – Not supported.
- HADR – Not supported.
- Synchronous Replication – Not supported.

3.14.15 Using IMRS Garbage Collection to Free Memory

There can be multiple IMRS garbage collector tasks in a database. Garbage collection in the context of row caching or snapshot isolation enabled-tables refers to reclaiming memory or data page space used for storing older versions.

The IMRS and LOB garbage collection threads run as background tasks, reclaiming memory or data page space in databases enabled for row caching or snapshot isolation.

By default, there are two IMRS garbage collector tasks for IMRS-enabled databases that are enabled with row storage. The garbage collector tasks wake up when the scanner or snapshot transaction having oldest timestamp completes.

Use the `number of imrs gc tasks per db` configuration parameter to specify the default number of IMRS garbage collector tasks that can be started when the server starts or when a IMRS-enabled database is brought online.

See the *Reference Manual: Configuration Parameters*.

You can increase the number of garbage collection tasks dynamically. However, decreasing the value of the configuration parameters requires you to restart the server to reduce the number of active garbage collection tasks.

Garbage collection for on-disk MVCC enabled databases collects `sysversions` entries and entries that were created as part of a transaction.

Using the `transaction snapshot` isolation level compared to the `statement snapshot` isolation level requires more space in the row storage cache and in `sysversions`. Transaction snapshot isolation retains the

complete versions of transactions that are executing because the transaction may request any of the active versions later in the transaction. However,during `statement snapshot` isolation level, the server frees up the memory asynchronously after a statement is complete and does not wait for the entire transaction to commit.

LOB cursors using the `statement snapshot` isolation level require more space for row storage cache and on disk versioning because the garbage collection does not release versions while the LOB cursor remains open. This occurs because the server does not materialize the LOB versions in the temporary database, and the cursor reviews all the versions if it needs to read the LOB column.

3.15 Troubleshooting the IMRS System

Generally, much of the troubleshooting procedures for the IMRS system include monitoring and managing available memory.

Use the `sp_imrs` system procedure to monitor the state of the IMRS cache, including the total amount of configured, used, and available memory, and the total number of inserted, migrated, and cached rows, and so on.

For example, this displays the cache information for the cache named `imrs_cache`:

```
sp_imrs show, cacheinfo, imrs_cache

CacheName  DBName TotalSizeMB UsedSizeMB FreeSizeMB PctUtil UsedSizeHWM
PctUtilHWM NumTables NumRows   NumRowsHWM NumVersions NumVersionsHWM
---------- ------ ----------- ---------- ---------- ------- -----------
---------- --------- --------- ---------- ----------- ---------------
imrs_cache tpcc     153600.21   90681.86   62918.35   59.03    98505.30
64.13          9 293702301  293921150        4489        1355491
```

This shows the row counts for the `imrs_cache`:

```
sp_imrs show, rowcounts, imrs_cache

DBName OwnerName ObjectName NRows     NRowsHWM  NVersions NVersHWM NInsRows
NMigRows NCachedRows NInsVers NMigVers NRowsPendGC
------ --------- ---------- --------- --------- --------- -------- ---------
-------- ----------- -------- -------- -----------
tpcc   [Any]     [Totals]   293700753 318922667      4856  1624820 274883592
16188017     1314572     4856        0        3970
tpcc   dbo       order_line 253894311 253894362         0   418518 253466448
427863           0        0        0           0
tpcc   dbo       orders      19241724  23058445      4856   217770
18007310        0     1234414     4856        0        3970
tpcc   dbo       stock       15860848  15870090         0   296821         0
15860848           0        0        0           0
tpcc   dbo       new_order    2587858   4056298         0   166730
2587858         0           0        0        0           0
tpcc   dbo       customer     1211238   1675534         0    31276         0
1211238           0        0        0           0
tpcc   dbo       history       821976  20249835         0   407658
821976         0           0        0        0           0
tpcc   dbo       item           80158    115463         0        0
0         0       80158        0        0           0
tpcc   dbo       district        2400      2400         0    58393         0
2400            0        0        0           0
tpcc   dbo       warehouse        240       240         0    27654
0      240           0        0        0           0
```

This displays the `imrslog` utilization:

```
sp_imrs show, sysimrslogs
Description
Value
Comment
-------------------------------------------------------------------------------
--------------------
-------------------------------------------------------------------------------
Total number of pages in
imrslogsegment
9216                    18.00
MB
Total number of pages in
use
1700                     3.32 MB
( 18.45 %)
Number of non-truncatable
pages
425                     0.83 MB
(  4.61 %)
Number of truncatable
pages
1275                     2.49 MB
( 13.83 %)
Total number of free pages of
imrslogsegment                                      7479
14.61 MB ( 81.15 %)
First page ID of sysimrslogs
14337

Last page ID of sysimrslogs
16043

Page ID of oldest non-truncatable page
15617

Percentage of space that can be freed by DUMP TRAN of total
space
75

Number of pages reserved for Last-Chance Threshold
(LCT)                          512                     1.00 MB
(  5.56 %)
(10 rows affected)(return status = 0)

Truncatable pages    : Number of pages between the first active page of
sysimrslog to the oldest insert transaction page.
Non-truncatable page : Number of pages between the oldest insert transaction
page to last active page of sysimrslog.
```

See *Reference Manual: Procedures* > *sp_imrs*.

3.15.1 Rectifying Lack of Memory in the IMRS Cache

There are a number of reasons a server configured for IMRS can run out of memory.

When the server runs out of memory, it issues a warning similar to this for updates of rows within the IMRS:

```
728 - Allocation of size 192 bytes from the row storage cache, 'imrs_cache',
failed as there is not enough memory available in the cache. Please increase the
cache size.
```

The transaction aborts if the server runs out of memory.

Seeing this error indicates that the server has an insufficient amount of IMRS memory available and that transactions will rollback. Error 728 is not raised for inserts and migrations that were transparently redirected to the page store due to insufficient memory in the IMRS cache. `update` and `delete` commands in versions earlier than 16.0 SP03 PL06 could also internally fall back and issue error 728. You can confirm this by querying the `NumPageStoreInsertsOOM` and `NumPageStoreUpdatesOOM` from the `monIMRSCacheActivity` monitoring table, which indicates the number of times a row was inserted or updated in the page store due to lack of memory, respectively. In this example, there are 19730222 inserts to the page store instead of the row store due to insufficient memory:

```
select NumPageStoreInsertsOOM, NumPageStoreUpdatesOOM from monIMRSCacheActivity
NumPageStoreInsertsOOM NumPageStoreUpdatesOOM
---------------------- ----------------------
19730222               0
```

See the subsequent sections for resolving 728 errors that are not handled internally. These actions are useful to avoid redirecting inserts and updates to the page store to keep hot data in memory.

IMRS Garbage Collection Threads not Keeping Up with Workload

This is caused when the server is configured for an insufficient number of IMRS garbage collection tasks to manage the garbage collection for the system. Symptoms include:

- Issuing `sp_who` indicates that the IMRS garbage collection threads are always running and never sleeping. You can see additional information in the `WaitStatus` column in the `sp_imrs show, gcinfo` output:

  ```
  WaitStatus
  ----------------
  runnable/running
  runnable/running
  ```

- The value in column `NTranPending` in the output of `sp_imrs show, gcinfo` is higher and increasing, and the value of `BlockingSPID` is changing across consecutive outputs, indicating that the number of transactions for which garbage collection is pending is growing and garbage collection threads are not blocked by a single, old SPID. For example:

  ```
  sp_imrs show, gcinfo
  DBName Type   Status   WaitStatus       ExitStatus LastWakeup
  BlockingSPID BlkSPIDType BlkSeconds NTranPending NTranBlocked
  ------ ------ -------- ---------------- ---------- -------------------
  ------------ ----------- ---------- ------------ ------------
  tpcc   imrsgc running  runnable/running alive      Feb  1 2017
  10:20AM          104 Transaction          0          10           0
  tpcc   imrsgc running  runnable/running alive      Feb  1 2017
  10:20AM          104 Transaction          0          10           0
  tpcc   imrsgc running  runnable/running alive      Feb  1 2017
  10:20AM          104 Transaction          0          10           0
  tpcc   imrsgc running  runnable/running alive      Feb  1 2017
  10:20AM          104 Transaction          0          10           0
  tpcc   lobgc  sleeping sleeping         alive      Feb  1 2017
  10:20AM            0 NULL                 0           0           0
  tpcc   lobgc  sleeping sleeping         alive      Feb  1 2017
  10:20AM            0 NULL                 0           0           0
  ```

- Issuing `sp_imrs 'show', 'rowcounts'` shows high and increasing values for `NRowsPendGC`. For example:

```
sp_imrs 'show', 'rowcounts'
DBName OwnerName ObjectName NRows     NRowsHWM  NVersions NVersHWM NInsRows
NMigRows NCachedRows NInsVers NMigVers NRowsPendGC
 ------ --------- ---------- --------- --------- --------- -------- --------
-------- ---------- -------- -------- -----------
 tpcc   [Any]     [Totals]   116914370 143885077     43368  3526929 94423383
22426891       32048    43368        0       24470
 tpcc   dbo       order_line  79909325  99944308         0  1266454 74152067
5742142       15116        0        0           0
 tpcc   dbo       orders      14712674  14876237     11674    30887
14069966   625783       16925    11674        0        6505
 tpcc   dbo       stock       14140324  14149941         0   203659        0
14140318           6        0        0           0
 tpcc   dbo       history      3244906   7969846     31694  1511008
3244906        0           0    31694        0       17965
 tpcc   dbo       new_order    2956444   3998022         0   413022
2956444        0           0        0        0           0
 tpcc   dbo       customer     1948056   2836850         0    34190        0
1948056           0        0        0           0
 tpcc   dbo       district        2400      2400         0    44352
0     2400           0        0        0           0
 tpcc   dbo       warehouse        240       240         0    23357
0      240           0        0        0           0
 tpcc   dbo       item               1    107233         0        0
0        0           1        0        0           0
```

If any of the situations described in the bullets above mentioned are true, the currently configured value for `number of imrs gc tasks per db` is insufficient and needs to be increased.

Old Open Transaction Blocking IMRS Garbage Collection from Freeing Versions

This occurs if there is a sufficient number of configured garbage collection threads, but they are blocked due to an old transaction. Issuing `sp_who` shows that the IMRS garbage collection threads are always sleeping, which can be verified with the `WaitStatus` column in output of `sp_imrs show, gcinfo`.

Issue `sp_imrs 'show', 'blocking_spid'` to determine the ID of the transaction blocking the threads. For example, this shows that spids 36 and 37 are preventing other threads from running:

```
sp_imrs 'show', 'blocking_spid'
 SPID UserName HostName    OSPID StartTime           NTransBlocked DBName  Status
 ---- -------- ----------- ----- ------------------- ------------- -------
----------
   36 dbo      linuxstore3 32441 Mar 29 2017  2:24AM             0 imrsdb  recv
sleep
   37 dbo      linuxstore3 6381  Mar 29 2017  2:27AM             0 imrsdb2 recv
sleep

To unblock GC of database imrsdb, execute the following -
sp_imrs clear, 'blocking_spid', 'imrsdb','36'
```

Issue the following command against the database to focus the investigation on. For example, this reviews the blocking spids in the `imrsdb2` database:

```
sp_imrs 'show', 'blocking_spid', 'imrsdb2'
```

```
SPID UserName HostName    OSPID StartTime           NTransBlocked DBName  Status
---- -------- ----------- ----- ------------------- ------------- -------
----------
  37 dbo      linuxstore3 6381  Mar 29 2017  2:27AM             0 imrsdb2 recv
sleep

To unblock GC of database imrsdb2, execute the following -
sp_imrs clear, 'blocking_spid', 'imrsdb2','37
```

To clear the blocking process, issue:

```
sp_imrs clear, 'blocking_spid', 'imrsdb','37'
Successfully cleared Blocking SPID 37 from the system.
```

Issue `sp_imrs` periodically without specifying the spid to clear any processes that might be blocking the IMRS GC. In the following example, the database `imrsdb` contains a blocking SPID 38:

```
sp_imrs clear, blocking_spid, 'imrsdb'
Successfully cleared Blocking SPID 38 from the system.
```

If there are no blocking processes, the command returns silently.

Pack Threads Are Not Keeping Up with the Workload

In this situation, issuing `sp_imrs` displays a lot of aggressive pack. In particular, `NAggrPct` shows the percentage of rows packed in aggressive mode, and `AggrTPct` shows the percentage of aggregated mode pack transactions in all pack thread transactions (both in bold below). For example:

```
sp_imrs 'show', 'pack_metrics'
Row counts for pack subsystem:
 CacheName  DBName NRows     InsPct MigPct CachedPct NDRPackPct NSteadyPct
NAggrPct SkipRatio NSkipped   HotRowsPct NoLockPct GT1VersPct OtherPct
 ---------- ------ --------- ------ ------ --------- ---------- ----------
-------- --------- ---------- ---------- --------- ---------- --------
 imrs_cache tpcc   228386737  50.30  49.40      0.20       0.00       8.20
91.50         5 1234787321       3.20      0.00      96.70     0.00
(1 row affected)
Total number of rows: 1
Memory related metrics for pack subsystem:
 CacheName  DBName LastPacked          LastVisited         PackedMB InsMBPct
MigMBPct CachedMBPct MemOvhdMB MemOvhdPct
 ---------- ------ ------------------- ------------------- -------- --------
-------- ---------- --------- ----------
 imrs_cache tpcc   Jan 19 2017  1:58AM Jan 19 2017  1:58AM    71191    43.60
56.00        0.20     24394      34.20
(1 row affected)
Total number of rows: 1
Pack efficiency and transaction metrics for pack subsystem:
 CacheName  DBName SkipRatio MemOvhdPct NCTrans DRTPct SteadyTPct AggrTPct
NRBTrans MBPerTran NRowsPerTran NDRowsPerTran NStRowsPerTran NAgRowsPerTran
 ---------- ------ --------- ---------- ------- ------ ---------- --------
-------- --------- ------------ ------------- -------------- --------------
 imrs_cache tpcc           5      34.20 1343690   0.00      28.80    71.10
1718         0          169          NULL             48            218
(1 row affected)
```

Sufficient, but Fragmented, Memory for IMRS Cache

Determine the minimum and maximum fragment size of the bucketpool corresponding to the IMRS cache with the `BucketPoolMaxFragSize` and `BucketPoolMinFragSize` columns of the `monBucketPool` table. In this example, the minimum fragment size is 32, and the maximum is 192. However, even though some buckets (including the global bucket) have some fragments to allocate, most of the larger-sized buckets do not include fragments. The memory available in the global bucket, below, is fragmented: the maximum fragment size is 192 bytes only, and with this configuration, any memory allocation request for more than this will fail:

```
select distinct BucketPoolName, BucketPoolMaxFragSize, BucketPoolMinFragSize
from master.dbo.monBucketPool where BucketPoolName="imrs_cache"
BucketPoolName                 BucketPoolMaxFragSize BucketPoolMinFragSize
------------------------------ --------------------- ---------------------
                    imrs_cache                   192                    32

select BucketPoolName, BucketSize, sum(InstanceNumFrags) as BucketNumfrags from
master.dbo.monBucketPool where BucketPoolName="imrs_cache" group by
BucketPoolName, BucketSize
go
BucketPoolName                  BucketSize BucketNumfrags
------------------------------ ----------- --------------
                    imrs_cache           0       70572027
                    imrs_cache          32          84710
                    imrs_cache          64        3200001
                    imrs_cache          96        3172138
                    imrs_cache         128        1525674
                    imrs_cache         160              0
                    imrs_cache         192              0
                    imrs_cache         224              0
                    imrs_cache         256              0
                    imrs_cache         288        3103354
                    imrs_cache         448              0
                    imrs_cache         800              0
                    imrs_cache         832              0
                    imrs_cache         864              0
                    imrs_cache        2560              0
                    imrs_cache       10944              0
                    imrs_cache       16448              0
                    imrs_cache       25824              0

                    imrs_cache   134217792              0
```

Insufficient Disk Space Causing Suboptimal Performance

When `imrslog` is 90% full, the server avoids inserting or migrating data into the IMRS, and operations are saved in the page store. The `imrs_ilm_dml_in_pgst_imrslog_availability` monitor counter reports the number of new DMLs processed in page store because `imrslog` is full;

```
select * from sysmonitors
where field_name = 'imrs_ilm_dml_in_pgst_imrslog_availability'
```

Once the space available in `imrslog` crossed the last-chance threshold (LCT, currently set to 5% for `imrslog`), all new transactions wait for log space to become available.

To prevent `sysimrslog` from running out of space:

- Run this command (which truncates the imrslog by the specified number of allocation units) followed by `dump tran` to truncate the `sysimrslog`:

```
dbcc imrs('maintain','imrslog',<'database_name>','move truncation point',
[#AU])
```

- Add additional `imrslog` disk space.

3.16 Performance and Tuning the IMRS

The IMRS cache uses the kernel's bucket memory manager internally, and should not require any external configuration, which is done directly by the IMRS memory manager.

However, it is a good policy to monitor the bucket manager:

- Monitoring the bucketpool level – issue `sp_imrs show, cacheinfo, imrs_cache` to view the IMRS cache usage, including the total size, allocated size, available size, and so on. Additionally, you can view this information from the `monBucketPool` monitoring table:

```
Bucketpool level information:
select distinct BucketPoolName, NumBuckets, NumInstances, Flags, NumSets,
BucketPoolUsed, BucketPoolUsedMax, BucketPoolOverhead, BucketPoolAllocs,
BucketPoolRetries, BucketPoolFailures, BucketPoolStats, BucketPoolOversize,
BucketPoolMaxFragSize, BucketPoolMinFragSize
from master.dbo.monBucketPool where BucketPoolName="imrs_cache"
 BucketPoolName                 NumBuckets  NumInstances Flags
NumSets     BucketPoolUsed       BucketPoolUsedMax    BucketPoolOverhead
BucketPoolAllocs BucketPoolRetries BucketPoolFailures BucketPoolStats
BucketPoolOversize   BucketPoolMaxFragSize BucketPoolMinFragSize
 ------------------------------ ----------- ------------ -----------
----------- -------------------- -------------------- --------------------
--------------- ----------------- ------------------ ---------------
-------------------- -------------------- --------------------
 imrs_cache                              20           64
2568           1          42438873728          41955303200
7400350528        410703071                0                  0
16384                   0            134311680                  32
```

- Monitoring the bucket level – view information about the different sizes of buckets, memory in buckets, the number of fragments in buckets, allocated, or freed, the number of times the server found a bucket empty, and the number of current non-empty instances.
  Use this information to get an general idea of the IMRS structure size distribution for the specified workload, including which size or bucket is hot, and which size or bucket is created but not used. Use the `found empty` counter as an indicator to seed those many fragments in a particular bucket.

```
select BucketPoolName, BucketSize, sum(InstanceNumFrags) as BucketNumfrags,
sum(InstanceAllocs) as BucketAllocs, sum(InstanceFrees) as BucketFrees,
sum(InstanceEmpty) as BucketFoundEmpty, BucketNonEmptyInsts
from master.dbo.monBucketPool where BucketPoolName="imrs_cache" group by
BucketPoolName, BucketSize
 BucketPoolName                 BucketSize  BucketNumfrags BucketAllocs
BucketFrees BucketFoundEmpty BucketNonEmptyInsts
 ------------------------------ ----------- -------------- ------------
----------- ---------------- ------------------
 imrs_cache                             160           4291     10668668
7072973         21768170                64
 imrs_cache                             160           4291     10668668
7072973         21768170                64
```

```
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              160           4291     10668668
7072973         21768170                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
 imrs_cache                              288         125182
205          92                0                  64
```

- Monitoring the instance level – view information about fragment distribution across instances per bucket, fragments skipped while searching through instance fragment list, and so on:

```
select BucketPoolName, BucketSize, InstanceID, InstanceNumFrags,
InstanceAllocs, InstanceFrees, InstanceFragSkipped
from master.dbo.monBucketPool where BucketPoolName="imrs_cache" and
BucketSize=32 and InstanceID=0
 BucketPoolName                 BucketSize  InstanceID InstanceNumFrags
InstanceAllocs InstanceFrees InstanceFragSkipped
 ------------------------------ ----------- ---------- ----------------
-------------- ------------- -------------------
 imrs_cache                              32          0
49973          17620         17590                  0
 imrs_cache                              32          0
49576          17982         17551                  0
 imrs_cache                              32          0
49702          17916         17615                  0
 imrs_cache                              32          0
49920          17616         17524                  0
 imrs_cache                              32          0
49413          18378         17789                  0
 imrs_cache                              32          0
49965          17255         17220                  0
 imrs_cache                              32          0
49935          17974         17909                  0
 imrs_cache                              32          0
50000          17919         17901                  0
 imrs_cache                              32          0
49834          18162         17990                  0
```

```
 imrs_cache                                32          0
49829          17766         17590                    0
```

There are a number of parameters you can configure to tune the bucket pool used for the IMRS cache, including:

- `maxfill` bucket pool value – this value indicates the maximum number of free fragments a server can hold. The `maxfill` bucket pool value is found in the `Bucket Pool:Worker Thread Memory` section of the server configuration file. New fragments are freed into the global bucket if the number of free fragments reaches this threshold. By default, `maxfill` is set to 50k per instance for IMRS cache. See *System Administration Guide > Setting Configuration Parameters > Bucket Pool Configuration Group Parameter* for more information about setting this parameter.
- `aggressive task stealing` – if a bucket is empty, and you enable stealing with the `aggressive task stealing` configuration parameter while allocating a fragment, the server attempts to steal or allocate a fragment from a higher sized bucket. By default stealing is enabled (set to 1).

3.17 Restrictions for IMRS

There are a number of commands, utilities, and features that are not supported for an IMRS-enabled environment.

- An IMRS-enabled database can have an `imrslog` (this is an on-disk row storage log, used for committing in-memory updates) or version storage (for storing on-disk row versions), but not both simultaneously. That is, SAP ASE supports either in-memory versioning (MVCC) or on-disk MVCC, but not both.
- Temporary databases cannot use a row storage cache.
- High availability
- Replication
- HADR
- Function-based indexes on IMRS-enabled tables.
- Using an IMRS as row-storage cache for IMDBs.
- Cumulative `dump` and `load` commands against IMRS-enabled database.
- Cross-platform dumps and loads of IMRS-enabled databases and transaction logs.
- Dumping and loading IMRS-enabled databases and transaction logs into a non-IMRS-enabled database.
- Loading an IMRS-enabled database into an archive database.
- You cannot issue alter database off to shrink data or log portions of IMRS-enabled databases.
- You cannot use proxy tables or proxy databases in an IMRS enabled database.
- You cannot issue mount or unmount against an IMRS-enabled database.
- These commands are not supported in an IMRS-enabled environment:
  - `create archive database`
  - `create existing table`
  - `create <proxy_table>`
  - `disk mirror`
  - `disk remirror`
  - `mount database`
  - `quiesce database`

- `transfer table`
- `unmount database`
- `create index with sorted_data` on a table that has some rows in the IMRS
- `create index with online`
- Issuing `disk reinit` or `disk refit` on imrslog devices
- `reorg rebuild with online`
- Partition-level commands (`split`, `move`, `merge`, `drop`, and `truncate partition` using the `with online` parameter).

4 Multiversion Concurrency Control

Multiversion concurrency control (MVCC) allows the server to lock rows for writing in one session while granting access to unaltered versions of these rows in another session. That is, reads never wait for writes, and writes never wait for reads.

SAP ASE offers two varieties of MVCC: Regular MVCC, which uses in-memory row versioning, where all inserts, updates, and deletes are performed in the in-memory row storage, and on-disk multiversion concurrency control (on-disk MVCC), which stores row versions in a temporary database specified when the database is created or altered. Multiversion concurrency control works the same regardless of the method you use (regular MVCC or on-disk MVCC). However, each version uses slightly different parameters to create or alter the database. For purposes of simplification, this description uses the term MVCC for both regular MVCC and on-disk MVCC unless explicitly stated otherwise.

MVCC enables concurrent `readwrite` operations. SAP ASE uses three isolation levels to enable multiversion concurrency control:

- `statement snapshot` – if a command is executed using the statement snapshot isolation level, the DML or scanner reads the data snapshot that existed before the start of the statement. All statements qualify the rows according to the timestamp of the statement. In this example, the scanner reads the value according to the snapshot that existed prior to the `insert` that occurs in session 2:

Session 1	Session 2

`1> select @@spid`
`2> go`
`------`
`21`
`1> use imrsdb`
`2> go`
`1> set tran isolation level`
`transaction snapshot`
`2> go`
`1> drop table t1`
`2> go`
`1> create table t1 (c1 int, c2 int)`
`lock datarows`
`2> go`
`1> insert into t1 values(1,1)`
`2> go`
`(1 row affected)`
`1> select * from t1`
`2> go`
`c1 c2`
`----------- -----------`
`1 1`	

	`1> select @@spid`
	`2> go`
	`------`
	`23`
	`1> use imrsdb`

Session 1	Session 2
	2> go 1> begin tran 2> go 1> set tran isolation level transaction snapshot 2> go 1> insert into t1 values(2,2) 2> go 1> select * from t1 2> go c1 c2 ----------- ----------- 1 1 2 2
1> select * from t1 2> go c1 c2 ----------- ----------- 1 1 (This session shows a single row of data because the second row was inserted after the transaction started.)	

- `transaction snapshot` – if a command is executed using the statement snapshot isolation level, the DML or scanner reads the data snapshot that existed before the start of the transaction. All statements within the transaction qualify the rows according to the timestamp of the transaction. The server requires additional row storage memory and space in `sysversions` if a transaction is executed using the transaction snapshot isolation level because all versions created by the transactions are valid and cannot be removed by the garbage collector. See Using IMRS Garbage Collection to Free Memory [page 90]. In this example, an open cursor grabs the locks, preventing the `update` from running:

Session 1	Session 2
1> select @@spid 2> go ------ 21 1> use imrsdb 2> go 1> drop table t1 2> go 1> create table t1 (c1 int, c2 int) lock datarows 2> go 1> insert into t1 values(1,1) 2> go (1 row affected)	

Session 1	Session 2
	1> select @@spid 2> go ------ 23 1> use imrsdb 2> go 1> begin tran 2> go 1> set tran isolation level transaction snapshot 2> go 1> declare cur1 cursor for select * from t1 for update 2> go 1> open cur1 2> go
1> update t1 set c1 = 4 2> go This command blocks because an open cursor took the lock. ROW LOCKS { 0x14da0a870 Row 0, Page 585, Ptnid 0, objid 720002565 , dbid 5, (bucket 2042) Local lockrecs off sw=0x0x14da0a910 ->lr=0x0x14da0b310 lrlid_floid=42 lrlid_tloid=42 ownerspid=21 lrtype=update row lrsemawait=0x0x14da0a910 lrstatus=(0x02 (LR_GRANTED)) lrscanindid=0x0 lrfileindex=394 (le_scrollop.cpp@@/main/11 2014-03-26) lrlinenum=556 Local lockrecs off sw=0x0x14da0b4a0 ->lr=0x0x14da0b040 lrlid_floid=46 lrlid_tloid=46 ownerspid=23 lrtype=update row lrsemawait=0x0x14da0b4a0 lrstatus=(0x00) lrscanindid=0x0 lrfileindex=749 (search.c@@/SAP_ASE/15 2014-10-23) lrlinenum=5728 Found 1 LOCKOBJs with 2 lock records hanging off of them.	

- `readonly statement snapshot` – if a command is executed using the readonly statement snapshot, scanners read the snapshot of data the existed before the start of statement, but `update` always updates the latest version of the row, regardless of the qualified version. In this example, the cursor reads version as per the timestamp when the cursor opens::

Session 1	Session 2
1> select @@spid 2> go ------ 21 1> use imrsdb 2> go 1> drop table t1 2> go 1> create table t1 (c1 int, c2 int) lock datarows 2> go 1> insert into t1 values(1,1) 2> go (1 row affected)	
	1> select @@spid 2> go ------ 23 1> use imrsdb 2> go 1> begin tran 2> go 1> set tran isolation level transaction snapshot 2> go 1> declare cur1 cursor for select * from t1 2> go 1> open cur1 2> go
1> update t1 set c1 = 4 2> go	
	1> fetch cur1 2> go c1 c2 ----------- ----------- 1 1 `fetch` returns a value of 1 and not a value of 4 because the value of `c1` was 1 when the cursor opened.

These isolation levels are in addition to the SQL standard isolation levels offered by SAP ASE, and are not part of SQL standard.

Snapshot isolation levels are enabled or disabled:

- At the session level for scans on the tables that have snapshot isolation enabled at the table- or database-level. The syntax is:

```
set transaction isolation level <isolation_level>
```

For example:

```
set transaction isolation level statement snapshot
```

- During a `select` statement. The syntax is:

  ```
  select ...
  from ...
  at isolation <isolation_level>
  ```

  For example:

  ```
  select title from titles
  where price > 10
  at isolation statement snapshot
  ```

The term "snapshot" implies that these isolation levels capture the state of the data at a particular time. Transactions commit only if the updates do not conflict with any concurrent update made since the snapshot.

- Scans or queries with isolation level set to `statement snapshot` and `readonly statement snapshot` acquire their timestamp at the start of the scan during query execution, and offer a transactionally consistent snapshot view of the data at the start of the query or scan.
- Scans or queries with isolation level set to `transaction snapshot` acquire their timestamp at the start of the first DML or scan with `transaction snapshot`, offering a transactionally consistent snapshot view of the data at the start of first DML or scan within a transaction in which the query or scan operates.

For each row altered by `update`, `delete`, or `insert` commands on snapshot isolation level-enabled tables, a new version of the row is created in the in-memory row storage or in the version store temporary database. The new version of the row receives a `commit` timestamp when the transaction that created the version commits. The `commit` timestamp in an ever-increasing number which reflects the commit sequence of any write transaction in a database: When a transaction commits, the `commit` timestamp is incremented.

All data row versions created by a transaction have a timestamp of infinity until the transaction commits. When the commit occurs, the row acquires a new unique timestamp, and all versions of data rows created by this transaction have this same timestamp.

Any `select`, `update`, or `delete` queries that execute simultaneously also receive a timestamp. The query that includes the datarows with the most recent timestamp qualifies as the latest version.

This examples shows a session without snapshot isolation enabled:

Session 1	Session 2	Comments
`select * from t1` Results returned from `select` statement	`begin tran update t1 set t1.a = 5 where t1.b = 'target'`	The scan for the `select` in session 1 blocks since the `update` in session 2 has acquired locks on some rows.
	`commit tran`	`select` in session 1 returns results only after session 2 commits.

If you enable snapshot isolation:

Session 1	Session 2	Comments
`set transaction isolation read committed snapshot`	`begin tran update t1 set t1.a = 5 where t1.b = 'target'`	The scan for the `select` in session 1 returns the results immediately and does not block, waiting for the `update` transaction in session 2 to complete
`select * from t1` Results returned from `select` statement	`commit tran`	

Multiversion Concurrency Control and Write-Conflicts

DMLs hit write conflicts if another transaction updates the version of a row on which the DMLs (updates, deletes, and so on) are operating.

How write conflicts are managed depends on which snapshot isolation level you are using:

- Statement isolation – the server reexecutes the statement with a newer timestamp. Use the `number of reexecutions` configuration parameter to determine how often the statement should reexecute.
- Transaction isolation – the server rolls back the statement with write conflict error (error number 12343).

Multiversion Concurrency Control and Locking

SAP ASE uses a "pessimistic" approach for locking at the snapshot isolation level. In this approach, updates and deletes acquire exclusive row locks on the data rows they modify or delete, but selects do not require locks. With multiversion concurrency control, updates are implemented by inserting new records, not by overwriting existing records, and transactions may write new versions of data while concurrent transactions retain `read` access to previous versions.

The pessimistic locking approach avoids the “lost update problem.” That is, situations where:

1. Transaction T1 updates, and then commits, a copy of a data row that is already stale.
2. An intermediate transaction, T2, (or a newer version of the data that T1 read) updates the data and then commits.
3. The update by transaction T2 is lost when transaction T1 commits.

Multiversion Concurrency Control and Cursors

Opening a cursor includes the following considerations in an in-memory row storage environment:

- Opening a `read` cursor on multiversion concurrency control- or on-disk MVCC-enabled tables using any snapshot isolation level materializes the affected rows. However, readonly statement snapshot isolation and ANSI isolation cursor do not materialize rows.
- Opening an `update` cursor on multiversion concurrency control- or on-disk multiversion concurrency control-enabled tables that use `statement snapshot` or `transaction` isolation levels materializes all affected rows and keeps an `UP_ROW` lock. The `UP_ROW` lock is upgraded to an `EX_ROW` lock if the row is updated using the cursor.
- Materializing rows requires disk space in the session's temporary database.
- LOB columns are not materialized in the worktable; instead, they are read from the user database. Because they are not materialized, a LOB cursor requires more space in the row storage cache and `sysversions` because all the versions will remain unchanged until the cursor is opened.
- Open cursor performs write-conflict checks on updatable cursors. If the cursor detects a write-conflict, open cursor fails at the transaction isolation level, but retries at the statement snapshot isolation level. Use the `number of reexecutions` configuration parameter to determine the number of times the statement reexecutes.
- If you declare a cursor without the `for read only` or `for update` parameters and open the cursor with snapshot isolation isolation level, it is treated as a `read only` cursor and updates and deletes are not allowed on this cursor. You must use the `for update` parameter when you create an `update` cursor using snapshot isolation level.

Configuring SAP ASE for Snapshot Isolation

Enable the `serializable snapshot isolation` or `read committed snapshot` at the query, session, or connection level.

Use the `create database` or `alter database` commands to indicate that all subsequent tables created in this database allow`snapshot_isolation` (unless you explicitly specify to not use `snapshot_isolation`).

You can configure SAP ASE to perform a scan using snapshot isolation at these levels:

- Database level – use `create database ... with snapshot_isolation` to create databases that use snapshot isolation. Use `alter database ... set snapshot_isolation` to alter an existing database to use snapshot isolation.
- Table level – use the `create table ... with snapshot_isolation` parameter to create a table that uses snapshot isolation. Use `alter table ... set snapshot_isolation` to alter an existing database to use snapshot isolation. You can use `create` and `alter table` commands to enable snapshot isolation on the table whether or not the database is configured for snapshot isolation, although the database must be enabled for `row_storage`.
- Query level – use the `select ... at isolation ...` parameter to explicitly specify the snapshot isolation level at the query level.
- Session level – use the `set transaction isolation level read committed snapshot | serializable snapshot` parameters.
- Connection level – see Connection Support for MVCC [page 123] for information about the `ISOLATION_MODE` connection-level property.

The default isolation level is `read committed`, and is only set to `read committed snapshot` or `serializable snapshot isolation` only if you specify this at the session level with the `set` command or at the query level with the `at isolation` parameter.

Utility Behavior During Snapshot Isolation Level

Consider these issues when you are issuing utilities and certain commands during snapshot isolation:

- Once the utility completes a transaction using transaction snapshot isolation level that was started before the operation finishes, the transaction hits error number 16865 if the affected table is referred. You may see this error multiple times if `shrinkdb_status` is executed in parallel.
- If a row has active versions, `reorg defrag` skips page defragmentation of the table in that allocation unit. The active versions may become obsolete after time, and these pages would be defragmented if a user runs `reorg defrag`.
- If an on-disk MVCC-enabled database is unmounted and then mounted on another SAP ASE server, the destination server must have an IMRS cache and a temporary database with same name, and neither the IMRS cache or the temporary database can be bound to another database.
- You cannot mount an on-disk MVCC-enabled database to versions of SAP ASE earlier than 16.0 SP03.
- BCP considerations:
  - Slow `bcp` – slow `bcp in` creates an insert version for each row being inserted by `bcp`. bcp provides the snapshot isolation for all the existing as well as new transactions because all the rows have versions, .
  - Fast `bcp` – fast `bcp` does not create row versions and instead, inserts the rows directly to pages. Because it does not create versions, it is possible that a row that was not visible when the snapshot scanner started, is later visible to the scanner. Although parallel snapshot transaction does not qualify the uncommitted rows as part of fast `bcp`, it is possible the scanner read some rows inserted by fast `bcp` if `bcp` finishes between the time the row was not visible and the time it appears.
  - You must enable the `sp_dboption select into/bulkcopy/pllsort` option to use fast `bcp`.

4.1 On-Disk Multiversion Concurrency Control

On-disk multiversion concurrency control (on-disk MVCC) stores row versions in a temporary database that you specify for row storage when you create or alter the database.

On-disk MVCC requires a memory-resident row storage cache to store links to row versions that reside in a temporary database of the version store. The row storage cache contains information about—and a link or pointer to—a data row version that resides in this temporary database, similar to the metadata used to describe and locate a row version.

For on-disk MVCC, the memory footprint overhead per version is 64 bytes. However, when a DML operation (for instance, update) triggers the migration of a row to the in-memory row storage (IMRS), the row version takes up 96 bytes of the IMRS cache. Long-running, and inactive transactions are preempted by the IMRS garbage collector, enabling the garbage collector to clear unused versions from the IMRS memory.

4.1.1 Estimating Required Resources for Enabling On-Disk MVCC

Before you enable on-disk MVCC, use the workload profiler to estimate the IMRS cache size and version storage size.

Prerequisites

Workload profiler is installed. See Managing the Workload with sp_wlprofiler [page 72].

Procedure

1. Run `sp_wlprofiler monitor` to start monitoring the database on which you want to enable on-disk MVCC. This example uses a database called `tpcc`:

```
sp_wlprofiler 'monitor', 'tpcc', "USING interval='00:02:00', feature='ODMVCC'"
go
Spid 49, Profiling Workload 1. The workload will be monitored for an interval
of '120' seconds after which the monitoring will 'end' ...
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_object' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_syslogs' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 43 rows to archive table 'sybdsamdb.dbo.WLP_monCachedObject' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_monDataCache' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceIO' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 3 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentIO' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 3 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentUsage'
with timestamp 'Jun  6 2018  8:15AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monOpenObjectActivity'
with timestamp 'Jun  6 2018  8:15AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monOpenPartitionActivity'
with timestamp 'Jun  6 2018  8:15AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTableCompression' with
timestamp 'Jun  6 2018  8:15AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTempdbActivity' with
timestamp 'Jun  6 2018  8:15AM'.
Sample MDA metrics, Workload ID 1, counter 100
Archived 51 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Jun  6 2018  8:16AM'.
Archived 17 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Jun  6 2018  8:16AM'.
Sample MDA metrics, Workload ID 1, counter 101
Archived 51 rows to archive table 'sybdsamdb.dbo.WLP_monProcessWaits' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 17 rows to archive table 'sybdsamdb.dbo.WLP_monSysWaits' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_object' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_spaceusage_syslogs' with
timestamp 'Jun  6 2018  8:17AM'.
```

```
Archived 43 rows to archive table 'sybdsamdb.dbo.WLP_monCachedObject' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 1 rows to archive table 'sybdsamdb.dbo.WLP_monDataCache' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceIO' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 3 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentIO' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 3 rows to archive table 'sybdsamdb.dbo.WLP_monDeviceSegmentUsage'
with timestamp 'Jun  6 2018  8:17AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monOpenObjectActivity'
with timestamp 'Jun  6 2018  8:17AM'.
Archived 7 rows to archive table 'sybdsamdb.dbo.WLP_monOpenPartitionActivity'
with timestamp 'Jun  6 2018  8:17AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTableCompression' with
timestamp 'Jun  6 2018  8:17AM'.
Archived 0 rows to archive table 'sybdsamdb.dbo.WLP_monTempdbActivity' with
timestamp 'Jun  6 2018  8:17AM'.
(return status = 0)
```

2. Monitor the workload using multiple iterations of the `monitor` parameter. All subsequent iterations of the `monitor` parameter use the same workload ID, interval, and sample time, and continue to collect metrics for the same monitoring session.

```
sp_wlprofiler 'monitor', 'tpcc'
```

3. Run `sp_wlprofiler finish` to stop monitoring on-disk MVCC features:

```
sp_wlprofiler 'finish'
go
Running plan for Workload Name='Workload-1' affecting target db = 'tpcc'
evaluating feature(s) 'ODMVCC'
Tables which are likely to benefit from feature 'ODMVCC' in Workload Name
'Workload-1' (ID = 1) affecting database 'tpcc' are listed below:
WLP_ID DBName Owner TableName     NRows     RowsInIMRS PctInIMRS IMRSCache_MB
VersionDb_MB
------ ------ ----- ------------- --------- ---------- --------- ------------
------------
      1 tpcc   dbo   item             100000          0      0.00
0.00         0.00
      1 tpcc   dbo   warehouse           240        240    100.00
1.52       196.29
      1 tpcc   dbo   district           2400       2400    100.00
2.04       196.29
      1 tpcc   dbo   new_order       3714544      30652      0.83
7.63       196.29
      1 tpcc   dbo   orders         25752248      30652      0.12
17.04       196.29
      1 tpcc   dbo   history        25354865      15326      0.06
21.11         0.00
      1 tpcc   dbo   customer        7200000      30652      0.43
129.16       196.29
      1 tpcc   dbo   stock          24000000     229893      0.96
386.11       392.58
      1 tpcc   dbo   order_line    256304302     459787      0.18
482.03       196.29
(9 rows affected)
Total IMRS Cache and Version DB sizes needed for the workload:
WLP_ID DBName IMRSCacheSizeMB VersionDbSizeMB
------ ------ --------------- ---------------
      1 tpcc              1811            1571
(1 row affected)
(return status = 0)
```

4. Issue the `report` parameter to report the estimated amount of IMRS cache and the required version database space:

```
sp_wlprofiler 'report'
go
Tables which are likely to benefit from feature 'ODMVCC' in Workload Name
'Workload-1' (ID = 1) affecting database 'tpcc' are listed below:
WLP_ID DBName Owner TableName     NRows     RowsInIMRS PctInIMRS IMRSCache_MB
VersionDb_MB
------ ------ ----- ------------- --------- ---------- --------- ------------
------------
      1 tpcc   dbo   item             100000          0      0.00
0.00         0.00
      1 tpcc   dbo   warehouse           240        240    100.00
1.52       196.29
      1 tpcc   dbo   district           2400       2400    100.00
2.04       196.29
      1 tpcc   dbo   new_order       3714544      30652      0.83
7.63       196.29
      1 tpcc   dbo   orders         25752248      30652      0.12
17.04      196.29
      1 tpcc   dbo   history        25354865      15326      0.06
21.11        0.00
      1 tpcc   dbo   customer        7200000      30652      0.43
129.16       196.29
      1 tpcc   dbo   stock          24000000     229893      0.96
386.11       392.58
      1 tpcc   dbo   order_line    256304302     459787      0.18
482.03       196.29
(9 rows affected)
Total IMRS Cache and Version DB sizes needed for the workload:
WLP_ID DBName IMRSCacheSizeMB VersionDbSizeMB
------ ------ --------------- ---------------
      1 tpcc             1811            1571
(1 row affected)
(return status = 0)
```

5. Issue the `report config` parameter to report configuration changes needed to enable the on-disk MVCC feature:

```
sp_wlprofiler 'report config'
go
List of configuration changes needed to enable feature 'ODMVCC' for the
tables in the database 'tpcc' seen in the workload 'Workload-1' (ID = 1):
/*
** BEGIN: ASE Configuration suggested by Workload Profiler for Workload
'Workload-1' (ID = 1) to enable feature 'ODMVCC'
**
** Configuration generated on : Mar 22 2018  5:21AM
**
** ASE Version: Adaptive Server Enterprise/16.0 SP03 PL04/EBF 27872 SMP/P/
x86_64/SLES 11.1/ase160sp03plx/0/64-bit/OPT/Wed Mar 21 23:40:20 2018
**
** Installwlprofiler version:
**
Script
Version
                     Status
-----------------
------------------------------------------------------------------------------
--------------------- --------
installwlprofiler 16.0 SP03 PL04/EBF 27872 SMP/P/x86_64/SLES 11.1/
ase160sp03plx/0/64-bit/OPT/Thu Mar 15 04:00:36 2018 Complete
(1 row affected)
*/
USE master
```

```
go
-- We need an additional 1811 MB of max memory to accomodate
-- the In-memory Row Storage Cache.
--
sp_configure 'max memory', 0, '296570M'
go
-- Create In-memory Row Storage Cache.
--
sp_cacheconfig 'tpcc_imrs_cache', '1811M', row_storage
go
-- Create the Version Storage device.
-- The physical location of the device needs to be updated
-- before running the command.
--
DISK INIT NAME = 'tpcc_version_db_dev'
        , PHYSNAME = '<full_path>/tpcc_version_db_dev'
        , SIZE = '1571M'
go
-- Create the Version DB
--
CREATE TEMPORARY DATABASE 'tpcc_version_db'
on 'tpcc_version_db_dev' = '1571M'
go
ALTER DATABASE tpcc VERSION STORAGE ON tpcc_version_db
go
ALTER DATABASE tpcc ROW STORAGE ON tpcc_imrs_cache
go
USE tpcc
go
ALTER TABLE dbo.item SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.warehouse SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.district SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.new_order SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.orders SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.history SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.customer SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.stock SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
ALTER TABLE dbo.order_line SET SNAPSHOT_ISOLATION ON USING VERSION STORAGE
go
/*
** END: ASE Configuration suggested by Workload Profiler for Workload
'Workload-1' (ID = 1) to enable feature 'ODMVCC'
*/
(return status = 0)
```

4.2 Scan and Query Behavior at Snapshot Transaction Isolation Levels

Scans at the `statement snapshot`, `transaction snapshot`, and `readonly statement snapshot` isolation levels use MVCC, and look for—and qualify—datarow versions with relevant time stamps.

For each `update`, `delete` or `insert` on snapshot isolation level-enabled tables, a new version of the row is created in the version store. This new version receives a `commit` timestamp when the transaction that created the version is committed.

Simultaneously, `select`, `update`, and `delete` queries execute with a timestamp. A query qualifies the latest versions of datarows on relevant tables with a `commit` timestamp that is less than its own timestamp. `select` queries execute with a timestamp at the `readonly statement snapshot` isolation level. The scans or queries with isolation levels `statement snapshot` and `readonly statement snapshot` acquire the timestamp at the start of the scan during each statement's query execution. Scans or queries with isolation level `transaction snapshot` acquire the timestamp at the start of the first DML or scan with this isolation level.

Based on the specified snapshot isolation level, the scans or queries look at the data that is most recently committed at the start of the transaction or query, offering transactionally consistent snapshot views of the data at a point in time.

The `statement snapshot` and `readonly statement snapshot` isolation levels offer a transactionally consistent snapshot of the data at the start of the query or scan.

The `transaction snapshot` isolation level offers a transactionally consistent snapshot of the data at the start of first DML or scan within a transaction in which the query or scan operates.

4.3 Using Snapshot Isolation Levels

The snapshot isolation levels affect only tables referenced in a query that are snapshot isolation-enabled tables, at the table or database level.

However, snapshot isolation levels are mapped to ANSI SQL-equivalent isolation levels according to the following table when you issue queries or DMLs on tables that are not enabled for snapshot isolation level scans, and you:

- Use the `set` command to configure the session for snapshot isolation levels.
- Use a `select` command to specify snapshot isolation levels.
- Issue a `select` or DML query on multiple tables, not all of which are MVCC-enabled.

Snapshot Isolation Level	Mapped to This SQL Isolation Level
`statement snapshot`	`read committed` (level 1)
`transaction snapshot`	`repeatable read` (level 2)

Snapshot Isolation Level	Mapped to This SQL Isolation Level
`readonly statement snapshot`	`read committed` (level 1)

Use these global variables to view information about isolation levels:

- `<@@isolation>` – returns the session-level isolation level setting.
- `<@@isolation_level_name>` – returns a string of the currently set session-level isolation level.
- `<@@snapshot_isolation>` – returns 0 if the session is not using snapshot isolation; returns 1 if the session is using snapshot isolation.

Isolation Level	@@isolation	@@isolation_level_name	@@snapshot_isolation
`read uncommitted`	0	Read uncommitted	0
`read committed`	1	Read committed	0
`repeatable read`	2	Repeatable read	0
serializable	3	Serializable	0
`statement snapshot`	11	Statement snapshot	1
`transaction snapshot`	12	Transaction snapshot	1
`readonly statement snapshot`	1	Readonly statement snapshot	1

4.4 Considerations when Using Snapshot Isolation Levels

Keep the following in mind when using snapshot isolation levels.

- Snapshot isolation is not available for statements compiled with compatibility mode.
- A `write` and `update` lock is acquired for `select ... for update` queries after qualifying the data row with the predicate. If write conflicts are detected during this scan, then for these isolation levels:
  - `statement snapshot` – the scan is internally restarted with a fresh timestamp.
  - `transaction snapshot` – the scan is aborted.
  - `readonly statement snapshot` – the `select ... for update` is be treated as DML and an ANSI isolation level is used since it acquires exclusive locks, eliminating the potential `write` conflicts.
- `readpast` clauses are silently ignored for `select` queries with snapshot isolation levels.
- If an `insert`, `select`, `update`, or `delete` statement involves multiple tables, some using snapshot isolation and some not, and transaction isolation level is set to one of the snapshot isolation levels through session level or query level setting, the server uses ANSI isolation levels for all the tables referenced in the statement, including MVCC-enabled tables. To fully benefit from snapshot isolation scans, all tables referenced in a statement should be MVCC-enabled.

- Because the query processor uses ANSI isolation levels if a query accessing multiple tables includes MVCC and non-MVCC-enabled tables, you should migrate all tables involved in referential integrity constraints to be MVCC-enabled.
- For bulk inserts in fast `bcp` mode, if there are concurrent scans that use statement or transaction snapshot isolation levels on the target table, `bcp` waits until the concurrent scans complete before it starts. Once `bcp` starts, it blocks any new snapshot isolation scans until it completes.
  For transactions that use the `transaction snapshot` isolation level, if some data rows are inserted into a MVCC-enabled table by `bcp`, scans in those transactions that start later stop with an error because the data rows inserted into the page store could cause inconsistency.
  In slow `bcp` mode, the data is inserted to the IMRS directly. The `bcp` utility and snapshot isolation scans will not block each other.
- Because you cannot perform a bulk insert and a snapshot scan on the same table, bulk inserts are disabled when you issue a `select .. insert` or `select ... merge` statement with the `enable bulk insert` configuration parameter enabled, and data is inserted into the IMRS.
- If your application is running in an MVCC-enabled environment, all tables accessed by application must be enabled for snapshot-isolation.

4.5 Using Snapshot Isolations on Tables with LOB Columns

There are additional considerations for off-row LOB data in MVCC-enabled tables.

MVCC snapshot isolation requires multiple versions of the old data. For data rows, old row versions are saved in the IMRS (in the case of sysversions in case of on-disk MVCC). However, LOB columns can hold up to 2 gigabytes of data, and storing several old versions of LOB data could consume a lot of memory space. Consequently, SAP ASE uses the page store to save old versions of LOB data. When off-row LOB data is deleted or updated, the old LOB data is not deleted; instead, it is marked as the old version and remains in the original database. When you issue an `update`, the new LOB data is saved in the page store in the same database, and a new data row version is created in the IMRS, with a text pointer indicating the new LOB data.

Locks are not acquired for scans using isolation levels to read data row versions in IMRS-enabled tables. Instead, the server uses the `transaction commit` timestamp associated with each data row version in the IMRS-enabled to determine if a data row is visible to the scanner. However LOB data versions saved in the page store do not include information about the `transaction commit` timestamp, so separate LOB version qualifications based on commit timestamp are not done for LOB data. Once version qualification is done for the data row, all LOB data belonging to this data row version is visible if it is visible to a scanner using snapshot isolation level.

Text pointers for LOB columns on MVCC-enabled tables contain the row ID (RID) of the home data row, instead of the number of first text page, which is used on tables that are not MVCC-enabled.

Considerations for individual commands on MVCC-enabled tables with LOB columns:

- `update` – if you are updating only non-LOB columns of an MVCC-enabled table, different in-memory versions of the row are created, with the LOB field containing the text pointer pointing to the same LOB data in the updated version of the row. However, if you are updating LOB columns of an MVCC-enabled table, then a different in-memory version of a row is created. A separate LOB page chain is created in the page store and the in-memory new version of the data row has a text pointer to this new version of LOB data in page store. Consequently, if you frequently update LOB columns, you will see increased disk space

consumption because old LOB versions require disk space until they are reclaimed by the background garbage collection task.

- `delete` – when a data row is deleted, the server creates a `delete` version of data row. Text pages of LOB columns belonging to this data row are not deallocated, but are marked `deleted`, and the LOB data remains in the page store. Therefore, you will not see decreased disk space consumption until the old LOB versions are reclaimed by background garbage collection task when you delete LOB column data.
- `readtext` – when run under snapshot isolation level, `readtext` attempts to generate a pointer to the correct version of LOB data (by using the correct data row version based on data row RID and the timestamp saved in the passed-in text pointer). However, if it cannot find a valid version of the data row, `readtext` fails and issues an error message.
  If you issue `readtext` on MVCC-enabled tables using ANSI isolation levels, it uses the text pointer saved in the latest committed data version of the data row to get and return LOB data. Shared data row locks and page locks on text pages are taken in this case, and are retained until the end of the transaction.
- `writetext` – when run under snapshot isolation level, `writetext` derives a pointer to the correct version of the LOB data (by using the correct data row version based on data row RID and the timestamp saved in the passed-in text pointer), the LOB data is successfully updated, and a new LOB version and data row version in the IMRS-enable table is created. However, it fails if:
  - The corresponding data row version is already collected by the garbage collection task.
  - The LOB version pointed to by the derived pointer is not the latest committed version, indicating a `write` conflict.

  When you specify `timestamp = sync_ts` (part of the OCS library for server connections), if the specified timestamp does not match the latest committed LOB version when snapshot isolation levels or conventional isolation levels are used, concurrently running tasks are updating the LOB data and creating the latest committed version prior to current task, and `writetext` stops with an error message.
  Exclusive data row locks are taken and held until the end of the transaction. Simultaneously, the server takes page locks on the text page, similar to running `writetext` on tables that are not MVCC-enabled.
- `drop table` and `truncate table` – when you drop or truncate a MVCC-enabled table with LOB columns, all LOB data, including old LOB versions, is cleaned up. You will see decreased disk usage after these commands finish.
- `alter table` – if only system catalogs must be updated when you alter an MVCC-enabled table with LOB columns, nothing will be done for the older LOB versions of the rows. However, the older LOB versions of the table must be cleaned up if the table requires data movement.
  If you have disabled `snapshot_isolation`, the server must clean all older LOB versions. If this occurs, you may see decreased disk space after the commands finish.
- `reorg rebuild` – older versions of LOB columns are cleaned up when you run `reorg rebuild` on MVCC-enabled tables with LOB columns. Other `reorg` commands (for example, `reorg defrag`) that do not touch LOB data and do not lock the target table in exclusive mode, do not clean older version of LOB columns: they are processed by the background garbage collection task.

Use the `number of lob gc tasks per db` configuration parameter to specify the default number of LOB garbage collector tasks that can be started when the server starts or when a IMRS-enabled database is brought online.

4.5.1 Off-Row and In-Row Data Movement

There are a number of items to consider for off-row and in-row data movement.

Off-row to in-row movement occurs when the `update` command updates LOB columns with in-row definitions. LOB data saved off row moves to in-row if the data row has enough space after updating in-row LOB columns in MVCC-enabled tables. In this case, `update` reclaims the disk space used by off-row LOB columns. For MVCC-enabled tables, the LOB data saved off-row becomes the old version (marked `versioned`), and a new data row version is created in the IMRS to save the new LOB data. The old LOB version is reclaimed by background garbage collection task.

The server moves data off-row after updating LOB columns with in-row definitions during in-row to off-row movement if the length of new LOB data is larger than the maximum allowed for an in-row length, or if there is not enough room in the data row. This typically happens when LOB columns are updated by `update` or `writetext` commands. For MVCC-enabled tables, if new LOB data does not fit into in-row storage, a new LOB version is created in the page store (off-row storage) directly, and a new version of the data row is created in the IMRS.

4.5.2 Reclaiming Pages Occupied by LOB Versions

When reclaiming pages occupied by LOB versions during runtime, each `update` or `delete` to a LOB column in MVCC-enabled tables creates an old LOB version in the page storage.

To reclaim disk space, hint information about these LOB versions is saved in memory in the IMRS, and a background system task checks each database and uses the hint information to release the disk space of old LOB versions when it is no longer needed.

A great number of old LOB data versions are generated after frequently updating or deleting LOB data in multiple databases. In this case, disk space used by old LOB data versions in some databases may not be released immediately because garbage collection task processes databases serially, and may be too busy processing old LOB versions in other databases.

When reclaiming pages occupied by LOB versions during startup, some hint information is saved in the page store during runtime whenever a LOB version is created, in addition to the hint information saved in memory. At startup, the hint information saved in memory no longer exists. The background system task finds each MVCC-enabled table in every database, and uses the saved hint information to reclaim disk space used by LOB versions created before the last server shutdown or crash.

If LOB versions are created after restarting the server from recent `delete` or `update` operations, these new LOB versions are processed by the background system task only after cleaning up the LOB versions created before the last server shutdown or crash. Consequently, if there are numerous LOB versions left before the server shutdown or crash, the new LOB version created after startup may not be processed immediately because the background system task may be too busy processing old LOB versions.

4.6 Creating and Altering Tables and Databases to Use Snapshot Isolation

The steps you take to create or alter tables and databases differ depending on whether they use row storage or on-disk MVCC.

4.6.1 Creating and Altering Tables for Row Storage MVCC

Create a row storage cache before creating or altering a table or database to use snapshot isolation for MVCC systems that use row storage.

Use the `create database` and `create table` commands to create databases and tables with snapshot isolation. The syntax is:

```
create database <database_name>
...
imrslog on <cache_name>
...
with snapshot_isolation {on | off}
```

And:

```
create table table_name
...
[row_caching {on [for {all} rows | off}]
...
with snapshot_isolation {on | off}
```

For example:

```
create database pubs2
on pubs2_dev1 = '900g'
log on pubs2_logdev1 = '200g'
imrslog on pubs2_imrs_logdev1 = '200g'
row storage on 'pubs2_row_cache'
with snapshot_isolation on
```

And:

```
create table currentpublishers (
      pub_id      char (4)        not null
    , pub_name    varchar (40)    null
    , city        varchar (20)    null
    , state       char (2)        null
)
with row_caching on, snapshot_isolation on
```

See Creating Databases that Use a Row Storage Cache [page 66], Creating Tables the Use Row Storage Cache and Deferred Allocation [page 69], and the *Reference Manual: Commands*.

Use the `alter database` and `alter table` commands to alter databases and tables to use snapshot isolation. The syntax is:

```
alter database database_name
```

```
. . .
set
   [, row_caching { on | off } [ for all tables ] ]
   [, snapshot_isolation { on | off } [ for all tables ]]
. . .
```

And:

```
alter table <table_name>
. . .
[, row_caching {on [for {all} rows] | off | default}]
. . .
[, snapshot_isolation { on | off } ]
. . .
```

For example:

```
alter database production
imrslog on prod_odrs_dev1 = "200g",
prod_odrs_dev2 = "100g"
snapshot_isolation on
```

And

```
alter table new_orders
set row_caching on,
snapshot_isolation on
```

See Dropping Tables and Indexes from the In-Memory Row Storage [page 71] and the *Reference Manual: Commands*.

4.6.2 Creating and Altering Databases and Tables for On-Disk MVCC

Create a row storage cache before creating or altering a table or database to use snapshot isolation for on-disk MVCC.

Creating Databases that Use On-Disk MVCC

> **i** Note
>
> The value for `max memory` must be sufficient for the cache size changes. If the value for `max memory` is insufficient, SAP ASE issues a message stating that you must increase this value. Disabling version storage or row storage behaves the same for on-disk MVCC databases, and both commands remove the row cache and version storage database.

To create a database that uses on-disk MVCC:

1. Create a named cache for the row storage usage:

   ```
   sp_cacheconfig “version_link_cache”, “50g”, row_storage
   ```

> **Note**
>
> On-disk MVCC requires space for version storage and for tracking versions. The version tracking table is located in the IMRS cache. How large a cache the MVCC requires depends on a number of factors, including the amount of changes to the tables, the length of time transactions are open, the number of concurrent users with different transaction times, statement versus transaction isolation, and so on.
>
> Because of the variety of dependencies, it is difficult to predict the precise size needed for the cache. One method is to use `sp_logging_rate` to estimate the size of the cache for intervals of 10 to 15 minute during peak periods and size the IMRS cache to the same size as the logging rate – assuming that most transactions commit normally within a 10-15 minute window (or any interval you decide).
>
> Minimally, the cache should be at least 2 GB in size. Once established, you can monitor `monIMRSCacheActivity` to right-size the cache and track statistics such as `MemForOlderVersions`, `MemForOlderVersionsHWM`, and so on.

2. Create a temporary database for the cache:

```
create temporary database mytempdb
```

3. Create devices for the database and log.
4. Create the database. The syntax is:

```
create [inmemory] [temporary] database <database_name>
...
[imrslog on database_device [= <size>] [, ...]]
[ row storage on cache_name [ FOR <number> ROWS ] ]
...
[[,] row_caching { ON | OFF }
...
[ <other sub-clauses> ... ]
```

For example:

```
create database production
on data_dev1 = '500g', data_dev2 = "500g"
log on log_dev1 = "200g", log_dev2 = "150g"
row storage on version_link_cache
version storage on mytempdb
with snapshot_isolation on using version storage
```

4.6.3 Altering Existing Databases to Use On-Disk MVCC

Use `alter database` to alter a database to use on-disk MVCC. This example uses the `pubs2` database.

Procedure

1. Create a row storage cache for `pubs2`:

```
sp_cacheconfig 'pubs2_row_cache', '300M', 'row_storage'
```

2. Create a temporary database for the cache:

```
create temporary database mytempdb
```

3. Alter the database to use the temporary database. The syntax is:

```
alter database <database_name>
[on {default | <database_device> [, ...] } ]
[log on <database_device> [,...] ]
[version storage on <tempdb_name> ]
```

   For example:

```
alter database pubs2
version storage on mytempdb
```

4. Add row storage to the `pubs2` database:

```
alter database pubs2
row storage on pubs2_row_cache
```

5. Alter `pubs2` to use snapshot isolation:

```
alter database pubs2
set snapshot_isolation on using version storage
```

4.6.4 Creating Tables that Use On-Disk MVCC

Tables that use version storage must be created in databases that are already configured for version link cache, and must use datarows locking.

Context

The syntax is:

```
create table table_name (column list> ...)
with snapshot_isolation on using version storage
```

Procedure

This example creates the `orders` table in the `pubs2` database. However, because the `model` database in this example uses allpages locking, the `create table` command requires the `lock datarows` parameter:

```
create table orders (
      id    int default NULL
    , price   money
    , name    varchar(30)
)
```

```
lock datarows
with snapshot_isolation on
using version storage
```

4.6.5 Altering Existing Tables to Use On-Disk MVCC

Use `alter table` to change a table to use version storage.

Context

The examples in this section use the `pubs2.titles` table.

Procedure

1. Make sure the table uses datarows locking to use the version link cache.
2. If it does not use datarowslocking, perform the following:

   ```
   alter table titles
   lock datarows
   ```

3. Drop and re-create all triggers on the table. For example, the `titles` table includes the `deltitle` trigger. Drop this trigger:

   ```
   drop trigger deltitle
   ```

4. Re-create the trigger (this is SQL from the `installpubs2` script):

   ```
   create trigger deltitle
   on titles
   for delete
   as
   if (select count(*) from deleted, salesdetail
   where salesdetail.title_id = deleted.title_id) >0
   begin
           rollback transaction
           print "You can't delete a title with sales."
   end
   go
   ```

5. Enable snapshot isolation on the table:

   ```
   alter table titles
   set snapshot_isolation on using version storage
   ```

4.7 Connection Support for MVCC

`ISOLATION_MODE` is a connection-level property that is used to enable the MVCC feature by converting ANSI isolation levels to snapshot isolation levels for MVCC- and on-disk MVCC-enabled tables.

> **i** Note
>
> SAP ASE provides the following techniques to migrate applications to use snapshot isolation seamlessly, and with minimal application changes. Perform these changes on the application in conjunction with any necessary server-side changes. If you do not set the `ISOLATION_MODE` property, existing applications default to run at ANSI isolation levels.

You can set the `ISOLATION_MODE` connection property for OCS, ODBC, and JDBC connections, and map ANSI isolation levels to snapshot isolation levels. When `ISOLATION_MODE` is set to `snapshot`, the isolation level maps as:

- Isolation level 0 and 1 – mapping to the `statement snapshot` isolation level
- Isolation level 2 and 3 – mapping to the `transaction snapshot` isolation level

Once the `ISOLATION_MODE` connection property is set, the server overrides the `set transaction isolation` command from the connection or system procedure, which avoids having to make application changes to set isolation levels. For example, if a stored procedure sets the isolation level to ANSI isolation levels 0, 1, 2, or 3, but when using MVCC-enabled tables and the `ISOLATION_MODE` property is set, the ANSI isolation level for the stored procedure is converted to the `statement snapshot` or `transaction snapshot` isolation levels described above.

`ISOLATION_MODE` can have these values:

- `default` – the applications use the isolation levels specified by the application.
- `snapshot` – the application uses the snapshot isolation levels listed in the following table when it specifies traditional ANSI isolation levels for MVCC-enabled tables.
- `readonly_snapshot` – the application uses the snapshot isolation levels listed in the following table when it specifies traditional ANSI isolation levels for MVCC-enabled tables.

Session-level Isolation Level	Connection Isolation Mode set to DEFAULT	Connection Isolation Mode set to SNAPSHOT	Connection Isolation Mode set to READONLY_SNAPSHOT
`read uncommitted`	`read uncommitted`	`statement snapshot`	`readonly statement snapshot`
`read commited`	`read commited`	`statement snapshot`	`readonly statement snapshot`
`repeatable read`	`repeatable read`	`transaction snapshot`	`transaction snapshot`
`serializable read`	`serializable read`	`transaction snapshot`	`transaction snapshot`

Session-level Isolation Level	Connection Isolation Mode set to DEFAULT	Connection Isolation Mode set to SNAPSHOT	Connection Isolation Mode set to READONLY_SNAPSHOT
`statement snapshot`	`statement snapshot`	`statement snapshot`	`statement snapshot`
`transaction snapshot`	`transaction snapshot`	`transaction snapshot`	`transaction snapshot`
`readonly statement snapshot`	`readonly statement snapshot`	`readonly statement snapshot`	`readonly statement snapshot`

CT-Lib Support

MVCC include options for the `ct_options()` CT-Library API routine.

- CS_OPT_ISOLATION_MODE supports:

```
CS_OPT_ISOLATION_DEFAULT
CS_OPT_ISOLATION_SNAPSHOT
CS_OPT_ISOLATION_READONLY_SNAPSHOT
```

- CS_OPT_ISOLATION supports:

```
CS_OPT_LEVEL0
CS_OPT_LEVEL1
CS_OPT_LEVEL2
CS_OPT_LEVEL3
CS_OPT_STATEMENT_SNAPSHOT
CS_OPT_TRANSACTION_SNAPSHOT
CS_OPT_READONLY_STATEMENT_SNAPSHOT
```

All options are supported by the OCS external configuration file (`ocs.cfg`). For example, setting options for `isql` in the `ocs.cfg` file looks similar to:

```
[isql]
CS_OPT_ISOLATION_MODE=CS_OPT_ISOLATION_READONLY_SNAPSHOT
CS_OPT_ISOLATION=CS_OPT_TRANSACTION_SNAPSHOT
```

JDBC Support

This example describes the JDBC support.

This example for JDBC maps `ISOLATION_MODE` uses:

- Property Name: ISOLATION_MODE
- Values: 0 / 1 / 2 (as described in the table above)
- 0: default
- 1: snapshot

- 2: readonly_snapshot

At runtime, the client application can set the `ISOLATION_MODE` property using this API:

```
connection.setClientInfo("ISOLATION_MODE","1");
connection.setClientInfo("ISOLATION_MODE","2");
```

The code is:

```
public static void main(String args[])
{
 Properties props = new Properties();
 props.put("user", "USER");
 props.put("password", "PASSWORD");
 // Client application can used MVCC feature
 props.put("ISOLATION_MODE", "1");
 // 1 = SNAPSHOT
 // 2 = READONLY_SNAPSHOT
 String url = "jdbc:sybase:Tds:HOST:PORT";
 Connection conn = null; conn = DriverManager.getConnection (url, props);
 // Dynamically changing isolation mode
 conn.setClientInfo("ISOLATION_MODE", "2");
 ...
 conn.setTransactionIsolation(SybConnection.TRANSACTION_STATEMENT_SNAPSHOT);
 ...
}
```

ODBC Support

This example describes the ODBC support.

The ODBC maps `IsolationMode` using the following values:

- Property Name: IsolationMode
- Values: 0 / 1 / 2 (as described in the table above)
- 0: default
- 1: snapshot
- 2: readonly_snapshot

You can specify the isolation mode either in the connection string or by using the `SQLSetConnectAttr` method:

```
int main()
{
// Code for declaring Environment and Connection handle
…
// Specifying Isolation Mode in connection string.
SQLDriverConnect(ConHandle, SQL_NULL_HANDLE, (SQLWCHAR*)L"driver=Adaptive Server
Enterprise;server=SERVER;port=PORT;UID=USER;PWD=PASSWORD;IsolationMode=1;", 255,
SQL_NULL_HANDLE, 0, SQL_NULL_HANDLE, SQL_DRIVER_NOPROMPT);
…
//Dynamically Specifying isolation mode using SQLSetConnectAttr() Method.
SQLUINTEGER isolation_mode = 1;
SQLSetConnectAttr(connection_handle, SQL_ATTR_TXN_ISOLATION_MODE,
(SQLPOINTER)isolation_mode, SQL_NTS);
…
}
```

Scripting Driver (Perl, PHP Python) Support

The `IsolationMode=` connection string property is supported for the Perl and PHP Python scripting drivers and maps to the `CS_OPT_ISOLATION_MODE` option. Valid values for this property are 0, 1 and 2.

This example uses Python:

```
conn = sybpydb.connect(dsn="User=sa;Password=;IsolationMode=0")
```

ADO.NET Support

This example describes the ADO.NET support.

The ADO.NET maps `IsolationMode` using the following values:

- Property Name: IsolationMode
- Values: 0 / 1 / 2 (as described in the table above)
- 0: default
- 1: snapshot
- 2: readonly_snapshot

In ADO.NET, you can only specify isolation mode in the connection string:

```
Public static void OpenConnectionToASE(){
//Specify isolation mode to "1" in connection string.
AseConnection conn_1 = new AseConnection("Data Source=SERVER;Port=PORT;User
ID=USER;pwd=SECRET;Database=DBName;IsolationMode=1;")
…
}
```

4.7.1 Using IsolationMode Property for ODBC Clients with DSN Connection

This section lists three scenarios describing how `IsolationMode` and `IsolationLevel` properties are used for ODBC clients in data source name (DSN) context.

- When an application uses a data source name (DSN) to specify a connection string and connect to a driver or a driver manager (DM), the application can specify isolation mode property (`IsolationMode`) in the DSN itself; no additional changes are required. In Microsoft Windows, you can specify it in the system DSN or as a user DSN in the system registry. For Linux, it's defined in the `odbc.ini` file.
- When an application does not use DSN but the connection string is hard-coded in the application itself, and the application connects to the driver or driver manager, change the connecting string in the application, then add the `IsolationMode` property to it.
- When an application changes the `IsolationMode` property dynamically using the `SQLSetConnectAttr()` API, the application functions when it is directly connected to the SAP ASE ODBC driver.

On Windows, do not use the default driver manager; this causes an error.
On Linux, you may also use UNIX ODBC.

When an application changes the `IsolationLevel` property dynamically to statement snapshot (`SQL_TXN_ASE_STATEMENT_SNAPSHOT`), transaction snapshot (`SQL_TXN_ASE_TRANSACTION_SNAPSHOT`), or readonly statement snapshot (`SQL_TXN_ASE_READONLY_STATEMENT_SNAPSHOT`) using the `SQLSetConnectAttr()` API, the application functions when it is directly connected to the SAP ASE ODBC driver.

On Windows, do not use the default driver manager; this causes an error.

On Linux, you may also use UNIX ODBC

4.8 Troubleshooting the On-Disk MVCC

Performance and tuning for the on-disk MVCC is closely tied to its IMRS cache and version storage resources.

Tuning the On-Disk Multiversion Concurrency Control

When the version storage resources are fully utilized, normal `insert`, `select`, `update`, and `delete` commands may be unable to proceed, and you may see errors similar to this when:

- The IMRS cache is full:

```
Msg 728, Level 17, State 7:
Line 1:
Allocation of size 2752 bytes from the row storage cache, 'imrs_cache', failed
as there is not enough memory available in the cache. Please increase the
cache
size. Memory allocator information = 0x10.
```

- The version database is full:

```
Msg 1105, Level 17, State 2:
Line 1:
Can't allocate space for object 'sysversions' in database 'version_tempdb'
because 'default' segment is full/has no free extents. If you ran out of space
in syslogs, dump the transaction log. Otherwise, use ALTER DATABASE to
increase
the size of the segment.
```

Monitor the current usage of the `sysversions` database using the `sp_spaceusage` or `sp_spaceused` procedures. Issue this to print the `sysversion` usage as the number of pages:

```
sp_spaceusage 'display using unit=pages', 'table', sysversions
All the page counts in the result set are in the unit 'pages'.
 OwnerName TableName   IndId NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
PctBloatUsedPages PctBloatRsvdPages
 --------- ----------- ----- ------- --------- --------- ---------- ------------
----------------- -----------------
 dbo       sysversions     0     0.0   62630.0  162703.0      38.49
8.0        2087566.67        2033687.50
```

Issue this to print the `sysversion` usage in the number of megabytes:

```
sp_spaceusage 'display using unit=MB', 'table', sysversions
All the page counts in the result set are in the unit 'MB'.
 OwnerName TableName   IndId NumRows UsedPages   RsvdPages    ExtentUtil
ExpRsvdPages PctBloatUsedPages PctBloatRsvdPages
 --------- ----------- ----- ------- ----------- ------------ ----------
------------ ----------------- -----------------
 dbo       sysversions     0     0.0 244.6484375 635.55859375      38.49       .
03125        2087566.67        2033687.50
```

Similarly, use `sp_imrs show` to monitor the current utilization of IMRS cache. For example:

```
sp_imrs show, cacheinfo, imrs_cache
 CacheName  DBName TotalSizeMB UsedSizeMB FreeSizeMB PctUtil UsedSizeHWM
PctUtilHWM NumTables NumRows NumRowsHWM NumVersions NumVersionsHWM
 ---------- ------ ----------- ---------- ---------- ------- -----------
---------- --------- ------- ---------- ----------- --------------
 imrs_cache tpcc      20480.24     256.00   20224.24    1.24      256.00
1.24         9       0          0           0              0
```

Cleansing On-Disk MVCC Resources

Resources for on-disk MVCC are consumed by either the latest or the oldest versions of the row. Each of these resources types are cleaned by a combination of the pack utility and IMRS GC threads.

- Latest row versions – the latest row versions consume memory from the IMRS cache. Query `monIMRSCacheActivity` to determine the IMRS cache usage for the latest row versions (sizes are in bytes):

  ```
  select MemForLatestVersions,MemForLatestVersionsHWM from monIMRSCacheActivity
  MemForLatestVersions MemForLatestVersionsHWM
  -------------------- -----------------------
                192192                 1679872
  ```

  Resources for the latest row versions are processed by the pack thread and sent to GC thread's queue, which frees this version of the row.
- Older row versions – older row versions consume memory from the IMRS cache and occupy space in the version storage database. Query `monIMRSCacheActivity` to determine the IMRS cache and version storage usage for older versions of rows:

  ```
  select
  MemForOlderVersions,MemForOlderVersionsHWM,NumSysversActivePages,NumSysversAct
  ivePagesHWM
  from monIMRSCacheActivity
   MemForOlderVersions  MemForOlderVersionsHWM NumSysversActivePages
  NumSysversActivePagesHWM
   -------------------- ---------------------- --------------------
  ------------------------
              381923072              1250692224
  435523                   1211363
  ```

  Resources for older versions are cleaned by IMRS GC thread. There may be a number of reasons for the older row version resources not being quickly cleaned. See Rectifying Lack of Memory in the IMRS Cache [page 92].

Pack Threads are not Packing Rows

This occurs if the threshold to trigger pack is set too high. In this situation, pack threads do not pack the latest row versions until the current IMRS cache utilization reaches this threshold. Symptoms for this are that pack-related counters (displayed by the `NumRowsPacked`, `MemPacked`, and `NumRowsSkippedbyPack` columns `monIMRSCacheActivity`) have zeroes as values, or do not change their values when queried successively:

```
select NumRowsPacked,MemPacked,NumRowsSkippedbyPack from monIMRSCacheActivity
 NumRowsPacked        MemPacked            NumRowsSkippedbyPack
 -------------------- -------------------- --------------------
                    0                    0                    0
```

To resolve this issue and cleanse the resources for the latest row versions, set the currently configured threshold to trigger pack to a lower value with the `imrs cache utilization` configuration parameter.

Pack Threads Are not Catching Up

This situation occurs if there is an insufficient number of pack tasks for the amount of pack work in the system. The symptoms for this situation are:

- The `MemForLatestVersions` and `MemForLatestVersionsHWM` pack counters from the `monIMRSCacheActivity` monitoring table are high and continuously increasing.
- The `MemPacked` and `NumRowsPacked` pack counters from the `monIMRSCacheActivity` monitoring table are high and continuously increasing.
- The `NumRowsSkippedbyPack` pack counter has a low value compared to `NumRowsPacked` in the `monIMRSCacheActivity` monitoring table.

For example:

```
select
MemForLatestVersions,MemForLatestVersionsHWM,MemPacked,NumRowsPacked,NumRowsSkipp
edbyPack from monIMRSCacheActivity
 MemForLatestVersions MemForLatestVersionsHWM MemPacked
NumRowsPacked        NumRowsSkippedbyPack
 -------------------- ---------------------- --------------------
-------------------- --------------------
           7265724502             7265787904         24277099206
126014113                5246
```

This situation indicates that resources for the latest row versions are not being cleaned quickly enough, even when the pack threads are working at their full capacity. You must increase the value of the `number of pack tasks per db` to increase the number of pack threads.

IMRS GC Threads Are not Keeping up with the Work

This situation occurs if there is an insufficient number of GC tasks, causing the pack threads to not pack the latest versions of the rows. The symptoms are that the:

- `MemForLatestVersions` and `MemForLatestVersionsHWM` pack counters from `monIMRSCacheActivity` are high and continuously increasing

- `NumRowsSkippedbyPack`, `NRowsSkippedPackGT1Versions`, `NRowsSkippedPackMisc`, and similar pack counters are high.

These conditions indicate that resources for the latest row versions are not getting cleaned quickly enough, and that pack threads are not packing the rows because:

- There is more than a single version of the row
- The latest version is not marked as `GC Ready`

Indicating the currently configured number of GC threads are not able to keep up with demand, causing the pack threads to not pack the rows.

To resolve this issue and cleanse the resources for the latest row versions, increase the number of IMRS GC threads with the `number of imrs gc tasks per db` configuration parameter.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP