

## In-Memory Database Option for Sybase® Adaptive Server® Enterprise

---

BY JEFFREY GARBUS, SOARING EAGLE CONSULTING

### INTRODUCTION

“Why can’t I just cache the whole database?”

“I don’t care about recovery, I want to turn off logging, I can’t afford the cost of the commits. I’ll restart everything in case of a failure.”

If I had a nickel for every time I’ve heard these requests over the years, I could buy a lot of coffee — even at today’s Starbucks prices.

There are some good, valid reasons for wanting to leverage cache and turn off logging. Caching a database provides a significant performance gain. If 100% of all activity is in memory (which might be 1000s of times faster than disk), you not only don’t have to worry about any physical IO (and the corresponding waits and process-swapping that go along with it), but you also have the benefit of reduced optimization time as Sybase Adaptive Server Enterprise (ASE) can skip the costing of IO buffering etc. This is all accomplished with the ASE in-memory database (IMDB) option.

What about logging? ASE has a write-ahead log. First ASE logs the changes it needs to make, then it makes the changes, then it logs the fact that the transaction is to be committed; when that is flushed to disk, the commit has completed successfully. This means if you are adding 20 gigabytes of data, you’re potentially really writing 40 gigabytes . For loads, in particular loading a data mart and/or warehouse or other reporting database, it is useful to be able to disable the commits (i.e. the physical writes to the transaction log). There are now 2 ways of doing this, by using either an IMDB or a relaxed-durability database (Rddb).

### A BIT ABOUT TRANSACTIONS

A transaction is an atomic unit of work. In simpler terms, any activity or combination of activities that needs to be performed as a unit may be bundled into a transaction. For example, when you transfer money from your checking account to your savings account using an ATM card, you want to know that in case of a sudden server crash, the money that came out of your checking account, and hadn’t yet made it to your savings account, finds its way back into the checking account. This is the purpose of a transaction.

From a more technical standpoint, transactions are supposed to pass the “ACID test ,” this acronym is used to describe transactional properties:

- Atomic
- Consistent
- Isolated
- Durable

IMDB and Rddb relax “durable” and “atomic” by partially or completely eliminating transaction logging and writes to disk. The result are significant performance gains.

## IN-MEMORY DATABASE

In-memory database (IMDB) is a technique used to place an entire database – data, logs, and all – in cache. This capability is a licensing add-on that enables you to treat memory-resident data the same way you treat any other ASE data (all T-SQL statements work). It's quick, easy, and integrated.

Because it is memory-resident, everything should be treated the same way you would treat tempdb -- the database disappears completely when the server is shut down.

There is no disk storage and no permanence whatsoever. In other words, "durable" has been sacrificed for performance. All IO is eliminated. Transaction logging is still enforced for rollbacks, triggers, and replication, but it is entirely in memory.

### When *NOT* to use IMDB

There's a reason for an "ACID test" for databases; they're built to house permanent data. You need to know, when your transaction commits, that you can later retrieve the information. This is where ASE shines.

If you need data permanence, and have no tolerance for data loss, use a disk-resident database (DRDB), which is ASE's default behavior.

### When to use IMDB

IMDB is best for situations where you don't care about data persistence and where performance is a priority. Ease-of-use is another benefit of an IMDB. It is accessed exactly the same as any other database; there's nothing new for developers to learn; business analysts simply need to remember that the data is ephemeral.

Use cases abound for IMDB. These include, but are not limited to:

- Frequently accessed look-up tables and cross-database or cross-server reference data
- Mass data loads and data cleansing have historically been performed in tempdb; you can now do that in a memory-resident database. You can even take this a step further by creating a memory-resident tempdb, create a tempdb group which includes this database, and place the data load user into that group (tempdb groups are also a feature of ASE 15.5)
- Data feeds can sometimes overwhelm physical storage, or transaction logs. Frequently customers have acquired solid-state disk (SSD) for the very purpose of managing transaction-log performance problems.
- Short-term, transient data, like e-commerce shopping carts, where the data (for the cart) is temporary, and persistence of the data is irrelevant until the order is placed

## RELAXED-DURABILITY DATABASE

Memory is less expensive today than ever before. However there are physical limitations. A 1-terabyte database may not fit in cache. Or, you may simply not have sufficient memory. There may be times where you want or need to create an amalgam of an IMDB and DRDB; hence, the relaxed-durability database (RDDB).

RDDB gives you the logging performance of an IMDB (i.e. doesn't write physical commits to disk), and the persistence of a DRDB; at least, as long as the system is shut down cleanly (if it is not, the database gets marked as suspect at startup).

The RDDB has two options, ephemeral and "mostly" permanent. The ephemeral option (`no_recovery`) means it gets recreated at startup, like the IMDB. The "mostly" permanent option ("`at_shutdown`") means that as long as the server is shut down nicely (i.e. it is not shut down with `nowait`), the database will remain available.

So, if it's RDDB rather than IMDB for size reasons, use `no_recovery` only. If it's RDDB rather than IMDB for persistence reasons, use "`at_shutdown`" as well.

## SETTING UP IMDB

Setting up an IMDB is remarkably similar to setting up any other database, with the additional step of predefining the cache.

- 1) Define the cache in which the database will exist
- 2) Create the devices in which the database will reside
- 3) Create the database

### Defining the database cache

The `sp_cacheconfig` stored procedure has been modified to accept an additional parameter, in order to define the cache as an IMDB cache (note there's no buffer replacement strategy here, as there is no FIFO queue in this database).

Syntax: `sp_cacheconfig 'Cache_Name', 'Cache Size', 'inmemory_storage'`

Example: `sp_cacheconfig 'Lookup DB Cache', '1G', 'inmemory_storage'`

### Creating the device

The disk init syntax has an additional type, to specify that it's an in-memory device.

Example:

`disk init name= 'LookUpDBDevice', physname = 'Lookup DB Cache', size = '1G', type = 'inmemory'`

It is just like creating any other device, you're just pointing the server towards a different resource from which to allocate pages.

### Creating the database

Database creation is similar, but there are a few additional options. Since the IMDB is recreated (in memory) every time the server starts, it has to come from a template. The default template is the model database (just like every other database). You can change this default to another template, as we're going to do below. Also as with model, any database options follow the database. Note further that as with other ASE objects, once a database becomes dependant on the template, the template can't be dropped until the dependant database is dropped.

Example:

`create inmemory database LookUpDB use ReferenceDB as template on LookUpDBDevice = '1G', log on LookUpLogDevice = '100M' with durability = no_recovery`

"inmemory" specifies that this database is not disk resident

"Durability = no\_recovery" specifies that the database will be recreated at startup.

"use ReferenceDB as template" specifies that the contents of the database will be loaded from ReferenceDB instead of from model.

This is a great way to preload a database into cache. We're creating an in-memory database called LookUpDB. At startup, it's going to create a 1G database (plus the log) in the cache specified by the LookUpDBDevice, and preload it with the contents of ReferenceDB, which may contain tables, stored procedures, data, and anything else that makes sense. It will be released at server shutdown.

## SETTING UP RDDB

There are two types of RDDB; ephemeral and persistent. The ephemeral database will be dropped and recreated at shutdown/startup regardless of the content. The persistent database, with its minimal logging, will only persist if the server is shut down cleanly (i.e. a “shutdown with nowait” will cause it to be marked as suspect at startup).

Example:

```
Create database MyRddb on rddbdev = 'zG' with durability = no_recovery
```

Or

```
Create database MyRddb2 on rddbdev2 = 'zG' with durability = at_shutdown
```

## MINIMALLY LOGGED DML OPERATIONS — AN IMPORTANT STEP

With ephemeral databases it is important to ensure that the server knows that logging is not a priority. Please note these commands ONLY work in an IMDB and RDDB database, they will be ignored in an DRDB database.

### Database-level DML logging

You can (and usually will) enable minimal-logging at the database level. This dramatically reduces log IO. There is no need to change anything at the application level.

You can turn this "on" at create database time, with the “dml\_logging = minimal” option, or later with the alter table option or off with the “dml\_logging = full” option.

### Table-level DML logging

This can also be turned off at the table-level with the create table or alter table statement:

```
Create table min_log (a int) with dml_logging = minimal
```

### Session-level DML logging

Session-level set options can be used if you want this to take effect only for your process:

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

## MAINTAINING IMDB/RDDB

Full syntax is available in the reference manuals; these databases may be altered as needed to extend their size dynamically.

## DUMP/LOAD

IMDB and RDDB databases can be dumped and loaded just like any other database, which offers additional possibilities for data population.

## TEMPDB

Tempdb databases can be an IMDB, Rddb, or DRDB. Moving forward, the type will most likely depend more on money than on anything else; this will make tempdb fly.

## SUMMARY: IMDB VS. Rddb VS. DRDB

IMDB databases are ephemeral; the need to be loaded at server startup, and they disappear at server shutdown. They are amazingly fast, because 100% of all activity is in memory.

DRDB databases (Sybase ASE's behavior for the past 20+ years) guarantees persistence of any data inserted into the database.

If the primary concern is performance, without regard to data persistence, use an IMDB. If data loss will not be tolerated due to server shutdown, use DRDB.

In between, there may be a situation where;

- 1) The IMDB database will not fit into cache, or
- 2) You have a persistent database with an IO bottleneck, and you are willing to potentially sacrifice the tail end of the database

IMDB has no durability. Rddb has optional durability, which is configurable at database definition time. DRDB is fully durable.

IMDB was just released, so we can't make formal recommendations based upon experience, but based on preliminary testing, I've got lots of plans:

- 1) Tempdb, where memory is available, is going into memory. I'm not even going to think hard about this one. In a high-performance environment, this will eliminate tempdb as a bottleneck (it always did for SSD, and this is a lot less expensive).
- 2) Lookup tables and other information that would typically go into its own cache are going into databases, with underlying databases as templates; this will "prime" cache, which we haven't been able to conveniently do before.
- 3) All data migration/load efforts are going to be performed in an IMDB if it fits in memory, and an Rddb if it doesn't.
- 4) I'm going to examine opportunities for IMDB, and look at high-availability options (Replication Server®, etc.) for continually pushing IMDB into a persistent database with acceptable latencies.

The use cases outlined above, and many more that spring to my mind (and yours), are getting due consideration for this easy-to-use answer to a lot of performance questions.

---

*A 20-year veteran of Sybase ASE database administration, design, performance, and scaling, Jeff Garbus has written over a dozen books, many dozens of magazine articles, and has spoken at dozens of user's groups on the subject over the years. He is CEO of Soaring Eagle Consulting, and can be reached at [jeff@soaringeagle.biz](mailto:jeff@soaringeagle.biz).*