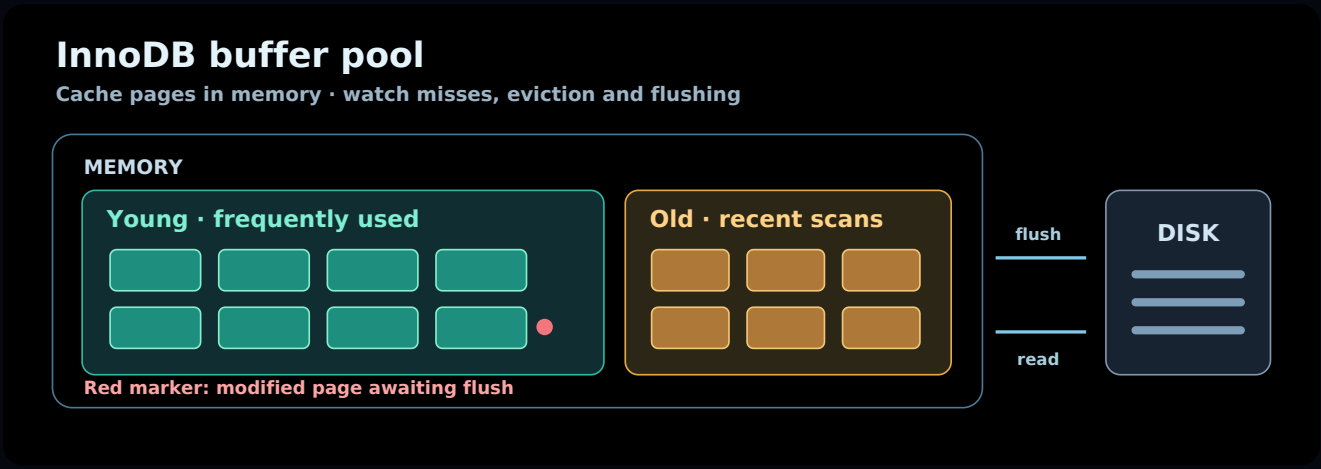
Understanding the InnoDB Buffer Pool: a guide to performance tuning

Duthaho · September 28, 2026

MYSQL INNODB BUFFER-POOL PERFORMANCE



InnoDB relies on memory to avoid reading every page from disk. Its buffer pool holds recently accessed data and index pages, and write operations modify pages there before they are flushed. Understanding what enters the pool, what leaves it, and how quickly it warms up helps explain both steady-state performance and slow queries after a restart.

What the buffer pool stores

`innodb_buffer_pool_size` sets the size of this shared memory area. InnoDB normally uses 16 KiB pages, although `innodb_page_size` can be configured differently when the instance is created. Data pages and index pages compete for room with internal structures. A page already in memory can be used without a data-file read; a missing page must be fetched from storage.

On a database-dedicated host, a large pool may be useful, but a fixed percentage of RAM is never a universal setting. Leave memory for the operating system, connections, temporary work, other engines and any co-located services. A 64 GiB host with a 48 GiB pool is an **example to test**, not a default recommendation.

Measure reads over the same interval, rather than relying on a lifetime percentage:

```
SHOW GLOBAL STATUS WHERE Variable_name IN (
  'Innodb_buffer_pool_read_requests',
  'Innodb_buffer_pool_reads',
  'Innodb_buffer_pool_pages_dirty',
  'Innodb_buffer_pool_wait_free'
);
```

The interval miss ratio is `ΔInnodb_buffer_pool_reads / ΔInnodb_buffer_pool_read_requests`. A rising ratio, more storage reads and worse query latency together suggest that the working set is no longer well served by memory. A high hit ratio alone does not prove that a query is fast. Dirty pages and `Innodb_buffer_pool_wait_free` add context about flushing pressure.

Why a large scan need not expel every hot page

InnoDB uses a modified least-recently-used list. Newly read pages enter its **old** section; frequently accessed pages can move into the **young** section. This midpoint insertion helps a one-off table scan avoid replacing the entire hot working set.

`innodb_old_blocks_pct` controls the share of the list reserved for the old section (37 by default in MySQL 8.4). `innodb_old_blocks_time` sets a delay before a newly loaded page can be promoted (1000 ms by default in MySQL 8.4). A longer delay can protect OLTP pages during large reports, but it can also keep genuinely useful pages in the old section. Setting it to zero removes that delay; it is not a general cure for read-heavy workloads. Change either setting only after measuring the real workload.

Warming the pool after a restart or failover

A restarted server begins with a cold cache. The first requests may spend more time reading data files, even when the SQL and indexes have not changed. InnoDB can save page **identifiers** at shutdown and reload those pages at startup using `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup`. `innodb_buffer_pool_dump_pct` controls how much of the hottest part is recorded. The dump does not contain the pages themselves; startup still performs disk reads.

Other choices are to let normal traffic warm the cache, run carefully bounded representative queries, or warm a replica before moving traffic to it. Preloading can compete with live traffic for

I/O, so use a rate limit and observe latency. A pool that is 80% full is not necessarily warm for the queries that matter. Judge readiness from the latency and disk reads of representative requests and, where applicable, the buffer pool load status.

How writes, redo and other structures interact

When a transaction changes a page, its cached copy becomes **dirty**. InnoDB writes redo records so recovery can replay committed changes, while background flushing eventually writes dirty data pages to disk. Flushing advances the checkpoint. The redo log and the buffer pool therefore serve different purposes: an aggressive dirty-page target can increase I/O, while too little flushing can create pressure later. Keep the durability setting appropriate to the workload; `innodb_flush_log_at_trx_commit=1` is the usual choice when every committed transaction must survive a server crash.

The **change buffer** can defer changes to non-unique secondary index pages that are not currently in memory. This can save random I/O on some write-heavy MySQL workloads, at the cost of later merges and memory use. The **adaptive hash index** can accelerate repeated lookups but consumes memory and may introduce contention. Neither feature is automatically beneficial: check whether the server version supports it, observe its counters and test with representative traffic before changing settings. In particular, MariaDB 10.5 and later use a single buffer pool instance, so advice to set `innodb_buffer_pool_instances=8` does not apply there.

A practical tuning sequence

For a busy order system, start with its real working set and a memory budget. Record query latency, physical reads, dirty pages and I/O at normal and peak load. If disk reads increase as the pool fills, improve indexes and query access patterns first, then test a larger pool while watching total host memory. If an analytical scan displaces hot pages, test the old-page settings. If latency spikes only after a restart or promotion, test dump/load and warm the future primary before routing production traffic.

On MySQL 8.4, multiple pool instances can reduce contention on multi-gigabyte pools, but their number should follow evidence and leave at least about 1 GiB per instance. A high-level cache-hit target such as 99% is a diagnostic hint, never a service-level objective on its own. Compare changes with the same workload and include both speed and memory or I/O regressions.

Conclusion

The buffer pool is more than a large cache. Its admission policy protects frequently used pages, warm-up determines post-restart latency, and flushing links in-memory writes to durable storage. Tune it with measured workload behavior and enough memory headroom rather than a universal configuration recipe.

Adapted and translated with the author's permission from Duthaho's article on Medium, originally published on 22 August 2025. Version-specific details were checked against the MySQL 8.4 buffer pool documentation and MariaDB buffer pool documentation.