

Want to read more?

You can [buy this book](#) at **oreilly.com**
in print and ebook format.

Buy 2 books, get the 3rd FREE!

Use discount code: OPC10

All orders over \$29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer,
including the iBookstore, the [Android Marketplace](#),
and [Amazon.com](#).



O'REILLY®

Spreading the knowledge of innovators

[oreilly.com](#)

High Performance MySQL, Third Edition

by Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko

Copyright © 2012 Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://my.safaribooksonline.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor: Andy Oram

Production Editor: Holly Bauer

Proofreader: Rachel Head

Indexer: Jay Marchand

Cover Designer: Karen Montgomery

Interior Designer: David Futato

Illustrator: Rebecca Demarest

March 2004: First Edition.

June 2008: Second Edition.

March 2012: Third Edition.

Revision History for the Third Edition:

2012-03-01 First release

See <http://oreilly.com/catalog/errata.csp?isbn=9781449314286> for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *High Performance MySQL*, the image of a sparrow hawk, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-31428-6

[LSI]

1330630256

Table of Contents

Foreword	xv
Preface	xvii
1. MySQL Architecture and History	1
MySQL's Logical Architecture	1
Connection Management and Security	2
Optimization and Execution	3
Concurrency Control	3
Read/Write Locks	4
Lock Granularity	4
Transactions	6
Isolation Levels	7
Deadlocks	9
Transaction Logging	10
Transactions in MySQL	10
Multiversion Concurrency Control	12
MySQL's Storage Engines	13
The InnoDB Engine	15
The MyISAM Engine	17
Other Built-in MySQL Engines	19
Third-Party Storage Engines	21
Selecting the Right Engine	24
Table Conversions	28
A MySQL Timeline	29
MySQL's Development Model	33
Summary	34
2. Benchmarking MySQL	35
Why Benchmark?	35
Benchmarking Strategies	37

What to Measure	38
Benchmarking Tactics	40
Designing and Planning a Benchmark	41
How Long Should the Benchmark Last?	42
Capturing System Performance and Status	44
Getting Accurate Results	45
Running the Benchmark and Analyzing Results	47
The Importance of Plotting	49
Benchmarking Tools	50
Full-Stack Tools	51
Single-Component Tools	51
Benchmarking Examples	54
http_load	54
MySQL Benchmark Suite	55
sysbench	56
dbt2 TPC-C on the Database Test Suite	61
Percona's TPCC-MySQL Tool	64
Summary	66
3. Profiling Server Performance	69
Introduction to Performance Optimization	69
Optimization Through Profiling	72
Interpreting the Profile	74
Profiling Your Application	75
Instrumenting PHP Applications	77
Profiling MySQL Queries	80
Profiling a Server's Workload	80
Profiling a Single Query	84
Using the Profile for Optimization	91
Diagnosing Intermittent Problems	92
Single-Query Versus Server-Wide Problems	93
Capturing Diagnostic Data	97
A Case Study in Diagnostics	102
Other Profiling Tools	110
Using the USER_STATISTICS Tables	110
Using strace	111
Summary	112
4. Optimizing Schema and Data Types	115
Choosing Optimal Data Types	115
Whole Numbers	117
Real Numbers	118
String Types	119

Date and Time Types	125
Bit-Packed Data Types	127
Choosing Identifiers	129
Special Types of Data	131
Schema Design Gotchas in MySQL	131
Normalization and Denormalization	133
Pros and Cons of a Normalized Schema	134
Pros and Cons of a Denormalized Schema	135
A Mixture of Normalized and Denormalized	136
Cache and Summary Tables	136
Materialized Views	138
Counter Tables	139
Speeding Up ALTER TABLE	141
Modifying Only the .frm File	142
Building MyISAM Indexes Quickly	143
Summary	145
5. Indexing for High Performance	147
Indexing Basics	147
Types of Indexes	148
Benefits of Indexes	158
Indexing Strategies for High Performance	159
Isolating the Column	159
Prefix Indexes and Index Selectivity	160
Multicolumn Indexes	163
Choosing a Good Column Order	165
Clustered Indexes	168
Covering Indexes	177
Using Index Scans for Sorts	182
Packed (Prefix-Compressed) Indexes	184
Redundant and Duplicate Indexes	185
Unused Indexes	187
Indexes and Locking	188
An Indexing Case Study	189
Supporting Many Kinds of Filtering	190
Avoiding Multiple Range Conditions	192
Optimizing Sorts	193
Index and Table Maintenance	194
Finding and Repairing Table Corruption	194
Updating Index Statistics	195
Reducing Index and Data Fragmentation	197
Summary	199

6. Query Performance Optimization	201
Why Are Queries Slow?	201
Slow Query Basics: Optimize Data Access	202
Are You Asking the Database for Data You Don't Need?	202
Is MySQL Examining Too Much Data?	204
Ways to Restructure Queries	207
Complex Queries Versus Many Queries	207
Chopping Up a Query	208
Join Decomposition	209
Query Execution Basics	210
The MySQL Client/Server Protocol	210
The Query Cache	214
The Query Optimization Process	214
The Query Execution Engine	228
Returning Results to the Client	228
Limitations of the MySQL Query Optimizer	229
Correlated Subqueries	229
UNION Limitations	233
Index Merge Optimizations	234
Equality Propagation	234
Parallel Execution	234
Hash Joins	234
Loose Index Scans	235
MIN() and MAX()	237
SELECT and UPDATE on the Same Table	237
Query Optimizer Hints	238
Optimizing Specific Types of Queries	241
Optimizing COUNT() Queries	241
Optimizing JOIN Queries	244
Optimizing Subqueries	244
Optimizing GROUP BY and DISTINCT	244
Optimizing LIMIT and OFFSET	246
Optimizing SQL_CALC_FOUND_ROWS	248
Optimizing UNION	248
Static Query Analysis	249
Using User-Defined Variables	249
Case Studies	256
Building a Queue Table in MySQL	256
Computing the Distance Between Points	258
Using User-Defined Functions	262
Summary	263

7. Advanced MySQL Features	265
Partitioned Tables	265
How Partitioning Works	266
Types of Partitioning	267
How to Use Partitioning	268
What Can Go Wrong	270
Optimizing Queries	272
Merge Tables	273
Views	276
Updatable Views	278
Performance Implications of Views	279
Limitations of Views	280
Foreign Key Constraints	281
Storing Code Inside MySQL	282
Stored Procedures and Functions	284
Triggers	286
Events	288
Preserving Comments in Stored Code	289
Cursors	290
Prepared Statements	291
Prepared Statement Optimization	292
The SQL Interface to Prepared Statements	293
Limitations of Prepared Statements	294
User-Defined Functions	295
Plugins	297
Character Sets and Collations	298
How MySQL Uses Character Sets	298
Choosing a Character Set and Collation	301
How Character Sets and Collations Affect Queries	302
Full-Text Searching	305
Natural-Language Full-Text Searches	306
Boolean Full-Text Searches	308
Full-Text Changes in MySQL 5.1	310
Full-Text Tradeoffs and Workarounds	310
Full-Text Configuration and Optimization	312
Distributed (XA) Transactions	313
Internal XA Transactions	314
External XA Transactions	315
The MySQL Query Cache	315
How MySQL Checks for a Cache Hit	316
How the Cache Uses Memory	318
When the Query Cache Is Helpful	320
How to Configure and Maintain the Query Cache	323

InnoDB and the Query Cache	326
General Query Cache Optimizations	327
Alternatives to the Query Cache	328
Summary	329
8. Optimizing Server Settings	331
How MySQL's Configuration Works	332
Syntax, Scope, and Dynamism	333
Side Effects of Setting Variables	335
Getting Started	337
Iterative Optimization by Benchmarking	338
What Not to Do	340
Creating a MySQL Configuration File	342
Inspecting MySQL Server Status Variables	346
Configuring Memory Usage	347
How Much Memory Can MySQL Use?	347
Per-Connection Memory Needs	348
Reserving Memory for the Operating System	349
Allocating Memory for Caches	349
The InnoDB Buffer Pool	350
The MyISAM Key Caches	351
The Thread Cache	353
The Table Cache	354
The InnoDB Data Dictionary	356
Configuring MySQL's I/O Behavior	356
InnoDB I/O Configuration	357
MyISAM I/O Configuration	369
Configuring MySQL Concurrency	371
InnoDB Concurrency Configuration	372
MyISAM Concurrency Configuration	373
Workload-Based Configuration	375
Optimizing for BLOB and TEXT Workloads	375
Optimizing for Filesorts	377
Completing the Basic Configuration	378
Safety and Sanity Settings	380
Advanced InnoDB Settings	383
Summary	385
9. Operating System and Hardware Optimization	387
What Limits MySQL's Performance?	387
How to Select CPUs for MySQL	388
Which Is Better: Fast CPUs or Many CPUs?	388
CPU Architecture	390

Scaling to Many CPUs and Cores	391
Balancing Memory and Disk Resources	393
Random Versus Sequential I/O	394
Caching, Reads, and Writes	395
What's Your Working Set?	395
Finding an Effective Memory-to-Disk Ratio	397
Choosing Hard Disks	398
Solid-State Storage	400
An Overview of Flash Memory	401
Flash Technologies	402
Benchmarking Flash Storage	403
Solid-State Drives (SSDs)	404
PCIe Storage Devices	406
Other Types of Solid-State Storage	407
When Should You Use Flash?	407
Using Flashcache	408
Optimizing MySQL for Solid-State Storage	410
Choosing Hardware for a Replica	414
RAID Performance Optimization	415
RAID Failure, Recovery, and Monitoring	417
Balancing Hardware RAID and Software RAID	418
RAID Configuration and Caching	419
Storage Area Networks and Network-Attached Storage	422
SAN Benchmarks	423
Using a SAN over NFS or SMB	424
MySQL Performance on a SAN	424
Should You Use a SAN?	425
Using Multiple Disk Volumes	427
Network Configuration	429
Choosing an Operating System	431
Choosing a Filesystem	432
Choosing a Disk Queue Scheduler	434
Threading	435
Swapping	436
Operating System Status	438
How to Read vmstat Output	438
How to Read iostat Output	440
Other Helpful Tools	441
A CPU-Bound Machine	442
An I/O-Bound Machine	443
A Swapping Machine	444
An Idle Machine	444
Summary	445

10. Replication	447
Replication Overview	447
Problems Solved by Replication	448
How Replication Works	449
Setting Up Replication	451
Creating Replication Accounts	451
Configuring the Master and Replica	452
Starting the Replica	453
Initializing a Replica from Another Server	456
Recommended Replication Configuration	458
Replication Under the Hood	460
Statement-Based Replication	460
Row-Based Replication	460
Statement-Based or Row-Based: Which Is Better?	461
Replication Files	463
Sending Replication Events to Other Replicas	465
Replication Filters	466
Replication Topologies	468
Master and Multiple Replicas	468
Master-Master in Active-Active Mode	469
Master-Master in Active-Passive Mode	471
Master-Master with Replicas	473
Ring Replication	473
Master, Distribution Master, and Replicas	474
Tree or Pyramid	476
Custom Replication Solutions	477
Replication and Capacity Planning	482
Why Replication Doesn't Help Scale Writes	483
When Will Replicas Begin to Lag?	484
Plan to Underutilize	485
Replication Administration and Maintenance	485
Monitoring Replication	485
Measuring Replication Lag	486
Determining Whether Replicas Are Consistent with the Master	487
Resyncing a Replica from the Master	488
Changing Masters	489
Switching Roles in a Master-Master Configuration	494
Replication Problems and Solutions	495
Errors Caused by Data Corruption or Loss	495
Using Nontransactional Tables	498
Mixing Transactional and Nontransactional Tables	498
Nondeterministic Statements	499
Different Storage Engines on the Master and Replica	500

Data Changes on the Replica	500
Nonunique Server IDs	500
Undefined Server IDs	501
Dependencies on Nonreplicated Data	501
Missing Temporary Tables	502
Not Replicating All Updates	503
Lock Contention Caused by InnoDB Locking Selects	503
Writing to Both Masters in Master-Master Replication	505
Excessive Replication Lag	507
Oversized Packets from the Master	511
Limited Replication Bandwidth	511
No Disk Space	511
Replication Limitations	512
How Fast Is Replication?	512
Advanced Features in MySQL Replication	514
Other Replication Technologies	516
Summary	518
11. Scaling MySQL	521
What Is Scalability?	521
A Formal Definition	523
Scaling MySQL	527
Planning for Scalability	527
Buying Time Before Scaling	528
Scaling Up	529
Scaling Out	531
Scaling by Consolidation	547
Scaling by Clustering	548
Scaling Back	552
Load Balancing	555
Connecting Directly	556
Introducing a Middleman	560
Load Balancing with a Master and Multiple Replicas	564
Summary	565
12. High Availability	567
What Is High Availability?	567
What Causes Downtime?	568
Achieving High Availability	569
Improving Mean Time Between Failures	570
Improving Mean Time to Recovery	571
Avoiding Single Points of Failure	572
Shared Storage or Replicated Disk	573

Synchronous MySQL Replication	576
Replication-Based Redundancy	580
Failover and Failback	581
Promoting a Replica or Switching Roles	583
Virtual IP Addresses or IP Takeover	583
Middleman Solutions	584
Handling Failover in the Application	585
Summary	586
13. MySQL in the Cloud	589
Benefits, Drawbacks, and Myths of the Cloud	590
The Economics of MySQL in the Cloud	592
MySQL Scaling and HA in the Cloud	593
The Four Fundamental Resources	594
MySQL Performance in Cloud Hosting	595
Benchmarks for MySQL in the Cloud	598
MySQL Database as a Service (DBaaS)	600
Amazon RDS	600
Other DBaaS Solutions	602
Summary	602
14. Application-Level Optimization	605
Common Problems	605
Web Server Issues	608
Finding the Optimal Concurrency	609
Caching	611
Caching Below the Application	611
Application-Level Caching	612
Cache Control Policies	614
Cache Object Hierarchies	616
Pregenerating Content	617
The Cache as an Infrastructure Component	617
Using HandlerSocket and memcached Access	618
Extending MySQL	618
Alternatives to MySQL	619
Summary	620
15. Backup and Recovery	621
Why Backups?	622
Defining Recovery Requirements	623
Designing a MySQL Backup Solution	624
Online or Offline Backups?	625
Logical or Raw Backups?	627

What to Back Up	629
Storage Engines and Consistency	632
Replication	634
Managing and Backing Up Binary Logs	634
The Binary Log Format	635
Purging Old Binary Logs Safely	636
Backing Up Data	637
Making a Logical Backup	637
Filesystem Snapshots	640
Recovering from a Backup	647
Restoring Raw Files	648
Restoring Logical Backups	649
Point-in-Time Recovery	652
More Advanced Recovery Techniques	653
InnoDB Crash Recovery	655
Backup and Recovery Tools	658
MySQL Enterprise Backup	658
Percona XtraBackup	658
mylvmbackup	659
Zmanda Recovery Manager	659
mydumper	659
mysqldump	660
Scripting Backups	661
Summary	664
16. Tools for MySQL Users	665
Interface Tools	665
Command-Line Utilities	666
SQL Utilities	667
Monitoring Tools	667
Open Source Monitoring Tools	668
Commercial Monitoring Systems	670
Command-Line Monitoring with Innotop	672
Summary	677
A. Forks and Variants of MySQL	679
B. MySQL Server Status	685
C. Transferring Large Files	715
D. Using EXPLAIN	719

E. Debugging Locks	735
F. Using Sphinx with MySQL	745
Index	771

MySQL Architecture and History

MySQL is very different from other database servers, and its architectural characteristics make it useful for a wide range of purposes as well as making it a poor choice for others. MySQL is not perfect, but it is flexible enough to work well in very demanding environments, such as web applications. At the same time, MySQL can power embedded applications, data warehouses, content indexing and delivery software, highly available redundant systems, online transaction processing (OLTP), and much more.

To get the most from MySQL, you need to understand its design so that you can work with it, not against it. MySQL is flexible in many ways. For example, you can configure it to run well on a wide range of hardware, and it supports a variety of data types. However, MySQL's most unusual and important feature is its storage-engine architecture, whose design separates query processing and other server tasks from data storage and retrieval. This separation of concerns lets you choose how your data is stored and what performance, features, and other characteristics you want.

This chapter provides a high-level overview of the MySQL server architecture, the major differences between the storage engines, and why those differences are important. We'll finish with some historical context and benchmarks. We've tried to explain MySQL by simplifying the details and showing examples. This discussion will be useful for those new to database servers as well as readers who are experts with other database servers.

MySQL's Logical Architecture

A good mental picture of how MySQL's components work together will help you understand the server. [Figure 1-1](#) shows a logical view of MySQL's architecture.

The topmost layer contains the services that aren't unique to MySQL. They're services most network-based client/server tools or servers need: connection handling, authentication, security, and so forth.

The second layer is where things get interesting. Much of MySQL's brains are here, including the code for query parsing, analysis, optimization, caching, and all the

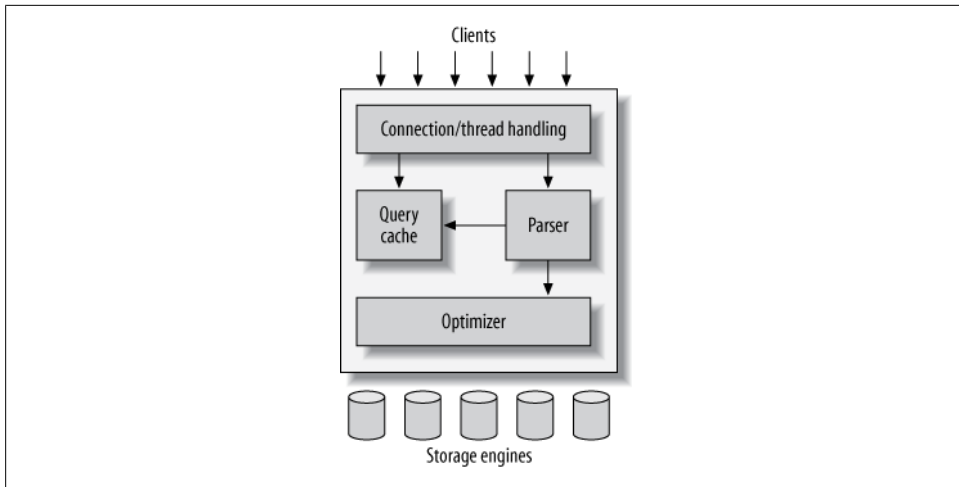


Figure 1-1. A logical view of the MySQL server architecture

built-in functions (e.g., dates, times, math, and encryption). Any functionality provided across storage engines lives at this level: stored procedures, triggers, and views, for example.

The third layer contains the storage engines. They are responsible for storing and retrieving all data stored “in” MySQL. Like the various filesystems available for GNU/Linux, each storage engine has its own benefits and drawbacks. The server communicates with them through the *storage engine API*. This interface hides differences between storage engines and makes them largely transparent at the query layer. The API contains a couple of dozen low-level functions that perform operations such as “begin a transaction” or “fetch the row that has this primary key.” The storage engines don’t parse SQL¹ or communicate with each other; they simply respond to requests from the server.

Connection Management and Security

Each client connection gets its own thread within the server process. The connection’s queries execute within that single thread, which in turn resides on one core or CPU. The server caches threads, so they don’t need to be created and destroyed for each new connection.²

When clients (applications) connect to the MySQL server, the server needs to authenticate them. Authentication is based on username, originating host, and password.

1. One exception is InnoDB, which does parse foreign key definitions, because the MySQL server doesn’t yet implement them itself.
2. MySQL 5.5 and newer versions support an API that can accept thread-pooling plugins, so a small pool of threads can service many connections.

X.509 certificates can also be used across an SSL (Secure Sockets Layer) connection. Once a client has connected, the server verifies whether the client has privileges for each query it issues (e.g., whether the client is allowed to issue a `SELECT` statement that accesses the `Country` table in the `world` database).

Optimization and Execution

MySQL parses queries to create an internal structure (the parse tree), and then applies a variety of optimizations. These can include rewriting the query, determining the order in which it will read tables, choosing which indexes to use, and so on. You can pass hints to the optimizer through special keywords in the query, affecting its decision-making process. You can also ask the server to explain various aspects of optimization. This lets you know what decisions the server is making and gives you a reference point for reworking queries, schemas, and settings to make everything run as efficiently as possible. We discuss the optimizer in much more detail in [Chapter 6](#).

The optimizer does not really care what storage engine a particular table uses, but the storage engine does affect how the server optimizes the query. The optimizer asks the storage engine about some of its capabilities and the cost of certain operations, and for statistics on the table data. For instance, some storage engines support index types that can be helpful to certain queries. You can read more about indexing and schema optimization in [Chapter 4](#) and [Chapter 5](#).

Before even parsing the query, though, the server consults the query cache, which can store only `SELECT` statements, along with their result sets. If anyone issues a query that's identical to one already in the cache, the server doesn't need to parse, optimize, or execute the query at all—it can simply pass back the stored result set. We write more about that in [Chapter 7](#).

Concurrency Control

Anytime more than one query needs to change data at the same time, the problem of concurrency control arises. For our purposes in this chapter, MySQL has to do this at two levels: the server level and the storage engine level. Concurrency control is a big topic to which a large body of theoretical literature is devoted, so we will just give you a simplified overview of how MySQL deals with concurrent readers and writers, so you have the context you need for the rest of this chapter.

We'll use an email box on a Unix system as an example. The classic *mbox* file format is very simple. All the messages in an *mbox* mailbox are concatenated together, one after another. This makes it very easy to read and parse mail messages. It also makes mail delivery easy: just append a new message to the end of the file.

But what happens when two processes try to deliver messages at the same time to the same mailbox? Clearly that could corrupt the mailbox, leaving two interleaved messages at the end of the mailbox file. Well-behaved mail delivery systems use locking to prevent corruption. If a client attempts a second delivery while the mailbox is locked, it must wait to acquire the lock itself before delivering its message.

This scheme works reasonably well in practice, but it gives no support for concurrency. Because only a single process can change the mailbox at any given time, this approach becomes problematic with a high-volume mailbox.

Read/Write Locks

Reading from the mailbox isn't as troublesome. There's nothing wrong with multiple clients reading the same mailbox simultaneously; because they aren't making changes, nothing is likely to go wrong. But what happens if someone tries to delete message number 25 while programs are reading the mailbox? It depends, but a reader could come away with a corrupted or inconsistent view of the mailbox. So, to be safe, even reading from a mailbox requires special care.

If you think of the mailbox as a database table and each mail message as a row, it's easy to see that the problem is the same in this context. In many ways, a mailbox is really just a simple database table. Modifying rows in a database table is very similar to removing or changing the content of messages in a mailbox file.

The solution to this classic problem of concurrency control is rather simple. Systems that deal with concurrent read/write access typically implement a locking system that consists of two lock types. These locks are usually known as *shared locks* and *exclusive locks*, or *read locks* and *write locks*.

Without worrying about the actual locking technology, we can describe the concept as follows. Read locks on a resource are shared, or mutually nonblocking: many clients can read from a resource at the same time and not interfere with each other. Write locks, on the other hand, are exclusive—i.e., they block both read locks and other write locks—because the only safe policy is to have a single client writing to the resource at a given time and to prevent all reads when a client is writing.

In the database world, locking happens all the time: MySQL has to prevent one client from reading a piece of data while another is changing it. It performs this lock management internally in a way that is transparent much of the time.

Lock Granularity

One way to improve the concurrency of a shared resource is to be more selective about what you lock. Rather than locking the entire resource, lock only the part that contains the data you need to change. Better yet, lock only the exact piece of data you plan to

change. Minimizing the amount of data that you lock at any one time lets changes to a given resource occur simultaneously, as long as they don't conflict with each other.

The problem is locks consume resources. Every lock operation—getting a lock, checking to see whether a lock is free, releasing a lock, and so on—has overhead. If the system spends too much time managing locks instead of storing and retrieving data, performance can suffer.

A locking strategy is a compromise between lock overhead and data safety, and that compromise affects performance. Most commercial database servers don't give you much choice: you get what is known as row-level locking in your tables, with a variety of often complex ways to give good performance with many locks.

MySQL, on the other hand, does offer choices. Its storage engines can implement their own locking policies and lock granularities. Lock management is a very important decision in storage engine design; fixing the granularity at a certain level can give better performance for certain uses, yet make that engine less suited for other purposes. Because MySQL offers multiple storage engines, it doesn't require a single general-purpose solution. Let's have a look at the two most important lock strategies.

Table locks

The most basic locking strategy available in MySQL, and the one with the lowest overhead, is *table locks*. A table lock is analogous to the mailbox locks described earlier: it locks the entire table. When a client wishes to write to a table (insert, delete, update, etc.), it acquires a write lock. This keeps all other read and write operations at bay. When nobody is writing, readers can obtain read locks, which don't conflict with other read locks.

Table locks have variations for good performance in specific situations. For example, `READ LOCAL` table locks allow some types of concurrent write operations. Write locks also have a higher priority than read locks, so a request for a write lock will advance to the front of the lock queue even if readers are already in the queue (write locks can advance past read locks in the queue, but read locks cannot advance past write locks).

Although storage engines can manage their own locks, MySQL itself also uses a variety of locks that are effectively table-level for various purposes. For instance, the server uses a table-level lock for statements such as `ALTER TABLE`, regardless of the storage engine.

Row locks

The locking style that offers the greatest concurrency (and carries the greatest overhead) is the use of *row locks*. Row-level locking, as this strategy is commonly known, is available in the InnoDB and XtraDB storage engines, among others. Row locks are implemented in the storage engine, not the server (refer back to the logical architecture diagram if you need to). The server is completely unaware of locks implemented in the