INSTANT

Short | Fast | Focused

InnoDB

A quick reference guide to walk you through the setup of InnoDB, and help you start unlocking the engine's potential

Matt Reid

[PACKT] PUBLISHING

Instant InnoDB

A quick reference guide to walk you through the setup of InnoDB, and help you start unlocking the engine's potential

Matt Reid

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

Instant InnoDB

First published: January 2013

Production Reference: 2240113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84968-748-5

www.packtpub.com

Cover image by Deborah Lattimore (deborah.lattimore@gmail.com)

Credits

Author
Matt Reid

Reviewers
Marc Delisle
Daniël van Eeden
Shiv Iyer

Acquisition Editor
Robin de Jongh

Commissioning Editor
Meeta Rajani

Technical Editors
Harsha Bharwani
Jalasha D'costa

Project Coordinators
Shraddha Bagadia
Esha Thakker

Proofreader
Mario Cecere

Indexer
Rekha Nair

Production Coordinators
Aparna Bhagat
Shantanu Zagade

Cover Work
Aparna Bhagat

About the Author

Matt Reid is a Systems Architect and Engineer who has focused his career on helping corporations solve large scale computing challenges. He is experienced in a wide range of Internet architectures ranging from global application load balancing to database clusters designed for serving millions of concurrent users.

His career has included work with geographically-distributed application and database environments, enterprise analytics and monitoring, database performance tuning, disaster recovery architectures, enterprise application programming, as well as systems infrastructure automation. He has worked and consulted for a multitude of companies including Riot Games, Adobe Software, British Sky Broadcasting, AdMob, Business Objects/SAP, OpSource, Arc90, Mattel, and NTT Communications.

Matt currently functions as the CTO and Engineering Architect for his consulting company MobileAxiom. His current projects include infrastructure automation engineering for the video game League of Legends, working on the global VPN architecture for startup Virtual Private Internet, and maintaining several open source projects for the MySQL community.

I would like to thank the following people for having an extraordinary impact on my career. Opportunity is everything and without you I would not be where I am today: Jim Reid, Deborah Lattimore, James Lester Reid, Molly Reid, Ryan Morgan, Monte Sanford, Dawn Baker, Don Barlow, Bruce Soto, Steve Crusenberry, and Dima Polyak. I'd also like to thank the Open Source community on the whole for helping to keep knowledge and intellectual resources free and transparent for the world.

About the Reviewers

Marc Delisle was awarded "MySQL Community Member of the Year 2009" because of his involvement with phpMyAdmin, a web interface for MySQL. He started to contribute to the project in December 1998, when he made the multi-language version. He is involved with phpMyAdmin as a developer, translator and project administrator, and enjoys meeting phpMyAdmin users in person.

Marc is a System Administrator at Cegep de Sherbrooke, Québec, Canada.

In addition to the *Mastering phpMyAdmin* successive editions, Marc has written *Creating your MySQL Database: Practical Design Tips and Techniques* and *phpMyAdmin Starter*, all with *Packt Publishing*.

Daniël van Eeden lives in the Netherlands. He works as a Database Consultant for Snow B.V. He blogs about databases at `http://databaseblog.myname.nl`. Besides working as a DBA, he has also worked as a System Administrator and Storage Administrator. Other areas of experience are High Availablity and Security. He is a member of the IOUG MySQL Council.

Shiv Iyer has over 12 years of experience in architecturing very large volume/ transaction intensive database applications in Oracle, SQL Server, MySQL, and PostgreSQL. Shiv specializes in Database Architecture, Performance Engineering, Scalability, High Availability, Disaster Recovery, Database Migration, Benchmarking, and Fine-Tuning and Database Security. He is the founder and Principal Database Architect of Ask DB Experts, a boutique "Database Technologies" consulting company based out of Bangalore, India.

Shiv has also worked on books such as *Oracle Database 11g Release 2 Performance Tuning Tips and Techniques* by *Richard Niemiec*, published by *McGraw-Hill Osborne Media*.

My parents, wife, family,friends, and Richard Niemiec

Table of Contents

Preface

InnoDB is one of the most commonly used database engines in the world today. Knowing how to effectively manage this technology can make or break a MySQL database environment. *Instant InnoDB* addresses everything you need to know for the installation, configuration, load testing, analysis, and long term support of an InnoDB-based database environment.

Explore in-depth features and performance tuning options for running the InnoDB database engine in production environments. *Instant InnoDB* contains detailed discussions for a range of simple to advanced topics. If you've ever wanted to learn more about InnoDB but didn't want to sit down with a dauntingly large or stale read then this reference guide is the answer to your queries. You will learn about common InnoDB monitoring and reporting applications as well as learn how to measure, analyze, and load test InnoDB workloads before going into production. This book also offers an in-depth explanation of configuration and tuning parameters in addition to troubleshooting and performance analysis methodologies.

What this book covers

Chapter 1, *Getting Started with InnoDB*, provides a quick overview of the core terminology and initial setup of the testing environment.

Chapter 2, *Basic Configuration Parameters*, will teach you about the most common settings and prerequisites for performance tuning.

Chapter 3, *Advanced Configuration Parameters*, covers advanced settings that can make or break a high performance installation of InnoDB.

Chapter 4, *Load Testing InnoDB for Performance*, explains all about general purpose InnoDB load testing as well as common methods for simulating production workloads.

Chapter 5, *Maintenance and Monitoring*, covers the important sections of InnoDB to monitor, tools to use, and processes that adhere to industry best practices.

Chapter 6, *Troubleshooting InnoDB*, explains all about identifying and solving common production issues that may arise.

Chapter 7, *References and Links*, can be referred to for informative data for further reading.

What you need for this book

To run the examples in the book, the following software will be required:

- Linux server:
    - Linux OS running Version 2.6.x or 3.x kernel
- MySQL applications:
    - MySQL Server 5.5 or later
    - MySQL Client 14.14 Distrib 5.5.x or later

Who this book is for

Instant InnoDB features content for all skill levels of MySQL administrators, developers, and engineers. Presented in an easy to read format, you will find answers to common questions and solutions to common problems, that will help you achieve higher performance and greater stability in your InnoDB based databases. If you've ever wondered how InnoDB manages memory buffers, how it can be used to scale from hundreds to thousands of queries per second, or how to monitor advanced analytics for an in-depth look at the health and performance during production use, then *Instant InnoDB* is the book for which you have been searching.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "InnoDB uses the `fsync()` call to flush data and logs to disk."

Any command-line input or output is written as follows:

```
mysql> create database test; use test;
Query OK, 1 row affected (0.00 sec)
Database changed
mysql> CREATE TABLE `test` (
    ->   `id` int(8) NOT NULL auto_increment,
    ->   `data` varchar(255),
    ->   `date` datetime,
    ->  PRIMARY KEY  (`id`),
    ->  INDEX `date_ix` (`date`)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.06 sec)
```

New terms and **important words** are shown in bold.

> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Getting Started with InnoDB

In the simplest terms, InnoDB is a transactional database information processing engine. It is the most commonly used engine currently available for the MySQL database application and was the first transactional engine offered with the application.

Different versions of MySQL ship with varying levels of InnoDB support and corresponding default settings for the InnoDB engine. Knowing which version of MySQL we're using will help us identify the baseline settings for the engine prior to going into development or production use of the database.

We'll cover the following topics:

- Everything needed to get up and running with the database as well as associated operating system requirements and supported hardware platforms
- How to ensure that InnoDB is available to MySQL and which default settings are enabled
- Basics of plugin configuration
- Basics of interacting with a new schema that will be used with InnoDB

Basic features of InnoDB

InnoDB is more than a fast disk-based relational database engine. It offers, at its core, the following features that separate it from other disk-based engines:

- MVCC
- ACID compliance
- Transaction support
- Row-level locking

These features are responsible for providing what is known as **Referential integrity**; a core requirement for enterprise database applications.

Referential integrity

Referential integrity can be best thought of as the ability for the database application to store relational data in multiple tables with consistency. If a database lacks consistency between relational data, the data cannot be relied upon for applications. If, for example, an application stores financial transactions where monetary data is processed, referential integrity and consistency of transactional data is a key component. Financial data is not the only case where this is an important feature, as many applications store and process sensitive data that must be consistent

Multiversion concurrency control

A vital component is **Multiversion concurrency control** (**MVCC**), which is a control process used by databases to ensure that multiple concurrent connections can see and access consistent states of data over time. A common scenario relying on MVCC can be thought of as follows: data exists in a table and an application connection accesses that data, then a second connection accesses the same original data set while the first connection is making changes to it; since the first connection has not finalized its changes and committed its information we don't want the second connection to see the nonfinalized data. Thus two versions of the data exist at the same time—multiple versions—to allow the database to control the concurrent state of the data. MVCC also provides for the existence of point-in-time consistent views, where multiple versions of data are kept and are available for access based on their point-in-time existence.

Transaction isolation

Transaction support at the database level refers to the ability for units of work to be processed in separate units of execution from others. This isolation of data execution allows each database connection to manipulate, read, and write information at the same time without conflicting with each other. Transactions allow connections to operate on data on an all-or-nothing operation, so that if the transaction completes successfully it will be written to disk and recorded for upcoming transactions to then operate on. However, if the sequence of changes to the data in the transaction process do not complete then they can be rolled back, and no changes will be recorded to disk. This allows sequences of execution that contain multiple steps to fully succeed only if all of the changes complete, and to roll back any changed data to its original state if one or more of the sequence of changes in the transaction fail. This feature guarantees that the data remains consistent and referentially safe.

ACID compliance

An integral part of InnoDB is its ability to ensure that data is atomic, consistent, isolated, and durable; these features make up components of ACID compliance. Simply put, **atomicity** requires that if a transaction fails then the changes are rolled back and not committed. **Consistency** requires that each successfully executed transaction will move the database ahead in time from one state to the next in a consistent manner without errors or data integrity issues. **Isolation** defines that each transaction will see separate sets of data in time and not conflict with other transactional data access. Finally, the **durability** clause ensures that any data that has been committed in a successful transaction will be written to disk in its final state, without the risk of data loss from errors or system failure, and will then be available to transactions that come in the future.

Locking characteristics

Finally, InnoDB differs from other on-disk storage engines in that it offers row-level locking. This primarily differs, in the MySQL world, with the MyISAM storage engine which features table-level locking. **Locking** refers to an internal operation of the database that prohibits reading or writing of table data by connections if another is currently using that data. This prevents concurrent connections from causing data corruption or forcing data invalidation when data is in use. The primary difference between table- and row-level locking is that when a connection requests data from a table it can either lock the row of data being accessed or the whole table of data being accessed. For performance and concurrency benefits, row-level locking excels.

System requirements and supported platforms

InnoDB can be used on all platforms on which MySQL can be installed.
These include:

- Linux: RPM, Deb, Tar
- BSDs: FreeBSD, OpenBSD, NetBSD
- Solaris and OpenSolaris / Illumos: SPARC + Intel
- IBM AIX
- HP-UX
- Mac OSX
- Windows 32 bit and 64 bit

There are also custom ports of MySQL from the open source community for running MySQL on various embedded platforms and non-standard operating systems.

Hardware-wise, MySQL and correspondingly InnoDB, will run on a wide variety of hardware, which at the time of this writing includes:

- Intel x86 32 bit
- AMD/Intel x 86_64
- Intel Itanium IA-64
- IBM Power architecture
- Apple's PPC
- PA-RISC 1.0 + 2.0
- SPARC 32 + 64 bit

Keep in mind when installing and configuring InnoDB, depending on the architecture in which it is installed, it will have certain options available and enabled that are not available on all platforms. In addition to the underlying hardware, the operating system will also determine whether certain configuration options are available and the range to which some variables can be set. One of the more decisively important differences to be considered while choosing an operating system for your database server is the manner in which the operating system and underlying filesystem handles write caching and write flushes to the disk storage subsystem. These operating system abilities can cause a dramatic difference in the performance of InnoDB, often to the order of 10 times the concurrency ability. These options will be explained in detail in *Chapter 3*, *Advanced Configuration Parameters*.

Downloading MySQL with InnoDB

MySQL should, unless you have a specific need for an older version, be downloaded directly from the MySQL website. Some operating systems maintain the latest version of MySQL in their package repositories while some do not, thus offering outdated and potentially insecure or unstable versions of the database application. Since end users do not have control over the operating system package repositories, and thus cannot upgrade the version of MySQL in the repository, it is professionally recommended that any server destined for production use be installed via the General Availability version of MySQL Community or Enterprise from the MySQL corporation website. The latest downloads can be found at `http://www.mysql.com/downloads/mysql/`.

As of MySQL 3.23, InnoDB has been included with the MySQL GA releases, and has been the primary choice for transactional engine support since. One of the more visible major changes came about in early versions of MySQL 5.1, when the InnoDB engine was shipped slightly different than before, where it was available as a pluggable storage engine. This new architecture, which allowed users to enable or disable storage engines on the fly, as opposed to compiling in support for different engines, made MySQL very flexible to different use cases. As such, using the plugin version of InnoDB allows the user to enable additional settings that were not available in the default included version of the engine.

As of MySQL 5.5, the default version of InnoDB is also the latest version of the plugin which negates the necessity of installing and enabling the formal plugin directly from InnoBase. From the MySQL website, the specific version differences are defined as follows:

> *As of MySQL 5.1.38, the InnoDB Plugin is included in MySQL 5.1 releases, in addition to the built-in version of InnoDB that has been included in the previous releases. MySQL 5.1.42 through 5.1.45 include InnoDB Plugin 1.0.6, which is considered of Release Candidate (RC) quality. MySQL 5.1.46 and up include InnoDB Plugin 1.0.7 or higher, which is considered of General Availability (GA) quality.*

Installing MySQL with InnoDB support

Since MySQL ships with the engine by default, the next step is to install MySQL. This can be achieved in different ways depending on your chosen architecture and platform. The scope of specific instructions for installation is not covered here due to the wide variety of platforms and architectures that MySQL supports.

Verifying InnoDB support and transaction settings

Knowing which version of InnoDB is available on your server is of prime importance when planning benchmarks, testing, production configuration, and associated performance tuning. As per the documentation, InnoDB has gained many variables to tune the performance and stability over the years. The first step to using InnoDB is to ensure that it is enabled and available on your MySQL server. You can check if InnoDB is enabled by running the following SQL queries from the MySQL command line or via another means of manual query execution such as MySQL Workbench or phpMyAdmin:

```
mysql> show engines\G
      Engine: InnoDB
```

```
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign
keys
Transactions: YES
          XA: YES
  Savepoints: YES
mysql> SHOW VARIABLES LIKE 'have_innodb';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_innodb   | YES   |
+---------------+-------+
1 row in set (0.00 sec)
```

Before creating any tables with InnoDB as the engine, it is important to check the setting for transaction isolation level. You can check the current setting of your running database by executing the following query:

```
[mysql://root@localhost/(none)]> SHOW
GLOBAL VARIABLES LIKE 'tx_isolation';

+---------------+-----------------+

| Variable_name | Value           |

+---------------+-----------------+

| tx_isolation  | REPEATABLE-READ |

+---------------+-----------------+

1 row in set (0.00 sec)
```

There are four options for this global configuration variable, and the setting determines the method in which InnoDB handles transactional data execution and locking. If the variable is set to one of the non-ACID settings when the application is expecting ACID ability from the database then this can result in dead locks, phantom data, inconsistent referential data, and even data loss.

ACID is defined by the following properties:

Atomicity

All transactions fail, or none of the transactions fail. Basically if a transaction fails because of a hardware issue, connection issue, and so on – those partial changes won't commit. It's 100 percent or 0 percent operation.

Consistency

Data being read by a `select` statement is all at the same state. So when you use a transaction you're getting the most current and consistent data available. This is related to multiversion concurrency control (MVCC).

Isolation

Nothing that's being read is actively being changed by another transaction. Your connection or transaction's read is not going to be changed by another transaction while you're dealing with that data.

Durability

Changes to the database persist – basically that means that if a transaction is committed and the DB fails or server crashes your changes will be there – which is why InnoDB uses transaction log files (where data is kept before being written to disk. The engine will read the logs on next startup and commit any remaining transactions in the logs that did not make to disk based tables).

Setting the transaction isolation level for InnoDB

The following options are available for this setting:

- **READ UNCOMMITTED**: Every `select` statement operates without locks so you don't get consistency and might have dirt reads, which are potentially earlier versions of data as defined by MVCC. As a result, this setting does not conform to ACID compliance standards and should never be used if your application issues transactions that require point-in-time consistent data reads.
- **READ COMMITTED**: This setting offers consistent reads without table or row locks. Each consistent read, even within the same transaction, sets and reads its own fresh snapshot of point-in-time data. Regardless, this setting does not conform to ACID compliance standards but it offers a compromise of consistency and performance for applications that do not require full ACID settings.

- **REPEATABLE READ**: The InnoDB default isolation level for ACID compliance. All reads within the same transaction will be consistent between each other – this defines the C in ACID. Additionally, all writes will be Durable, all transactions Atomic, all reads Isolated. Unless you have specific reasons to do otherwise, this should be your default setting for the transaction-isolation configuration variable.
- **SERIALIZABLE**: This is the same as REPEATABLE READ but MySQL converts regular `select` statements with the preface of LOCK IN SHARED MODE when auto-commit is enabled. If auto-commit is disabled then each `select` statement is started in a separate transaction, which will ensure that all reads are consistent. This setting also allows for XA distributed transactions support, which you can read more about in the MySQL manual. The SERIALIZABLE value setting will impact database transaction execution performance, so only enable this if it is absolutely necessary.

Creating your first InnoDB table

Once InnoDB support has been verified, you can start using the engine in your table definitions. A simple test to interact with InnoDB can be run as follows, where we will create a table with the engine specified explicitly along with a primary key, which InnoDB uses to index data.

If you leave off the engine definition at the end of the table create statement then MySQL will create the table with the system default engine, which is defined by the startup variables: `default-storage-engine` or, alternately, `storage_engine`, both accomplish the same purpose.

If you plan to use InnoDB exclusively for your table engine definitions, it generally makes sense to ensure that tables are created, explicitly and implicitly, by changing one or both of those startup variables to InnoDB. As of MySQL Version 5.5, the default has been changed to InnoDB so depending on your version you may not need to explicitly define the variable.

```
mysql> create database test; use test;
Query OK, 1 row affected (0.00 sec)
Database changed
mysql> CREATE TABLE `test` (
    ->   `id` int(8) NOT NULL auto_increment,
    ->   `data` varchar(255),
    ->   `date` datetime,
    ->  PRIMARY KEY  (`id`),
    ->  INDEX `date_ix` (`date`)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.06 sec)
```

In the preceding table create statement, we created one table with three columns. These columns are described as follows:

- A primary key based `id` field that will automatically increase in value for every inserted row
- A variable character based `data` field to store our values
- `Datetime` based `date` field to record the insert date

Besides edge cases, these columns can be considered the bare minimum columns for relational data. The automatically incremented `id` field will allow InnoDB to efficiently index our data or rapid lookups, and the `date` field will allow us to easily sort and search our data based on the time it was created in the table.

These two columns are specifically important for InnoDB based tables, as other table engines can operate relatively quick and efficient full table scans without a primary key column if running a typical query such as `SELECT COUNT(*) FROM test;`. However, InnoDB requires a PRIMARY KEY column or other INDEX to read in order for the query results to remain expedient; thus our `id` column provides that performance gain. Instead of writing the inefficient latter query, we can now write `SELECT COUNT(id) FROM test;` and InnoDB will access the `id` column PRIMARY KEY index which results in a very fast result versus a full table scan. Similarly, we can sort our table data via the `date` column and get quickly-returned table data.

If you'd like to experiment with InnoDB features more easily or see how different versions of MySQL work with InnoDB, you can install a sandboxing tool such as MySQL Sandbox.

Summary

At this point you should know the hardware and operating system requirements for installing MySQL, as well as the basic process for downloading and enabling support for the InnoDB engine. We've covered the requirements for ACID compliance and shown you how to set up basic indexing, as well as the process for creating a sample InnoDB table. Next we'll get into the basics of tuning InnoDB for better performance.

2

Basic Configuration Parameters

When reading the MySQL documentation you may find that InnoDB has over fifty-eight configuration settings, more or less depending on the version, for tuning the performance and operational defaults. The majority of these default settings can be left alone for development and production server environments. However, there are several core settings that can affect great change, in either positive or negative directions depending on the application workload and hardware resource limits, with which every MySQL database administrator should be familiar and proficient.

Keep in mind when setting values that some variables are considered dynamic while others are static; dynamic variables can be changed during runtime and do not require a process restart while static variables can only be changed prior to process start, so any changes made to static variables during runtime will only take effect upon the next restart of the database server process. Dynamic variables can be changed on the MySQL command line via the following command:

```
mysql> SET GLOBAL  [variable]=[value];
```

If a value is changed on the command line, it should also be updated in the `global my.cnf` configuration file so that the change is applied during each restart.

MySQL memory allocation equations

Before tuning any InnoDB configuration settings—memory buffers in particular—we need to understand how MySQL allocates memory to various areas of the application that handles RAM. There are two simple equations for referencing total memory usage that allocate memory based on incoming client connections:

- **Per-thread buffers**: Per-thread buffers, also called per-connection buffers since MySQL uses a separate thread for each connection, operate in contrast to global buffers in that per-thread buffers only allocate memory when a connection is made and in some cases will only allocate as much memory as the connection's workload requires, thus not necessarily utilizing the entire size of the allowable buffer. This memory utilization method is described in the MySQL manual as follows:

  > *Each client thread is associated with a connection buffer and a result buffer. Both begin with a size given by net_buffer_length but are dynamically enlarged up to max_allowed_packet bytes as needed. The result buffer shrinks to net_buffer_length after each SQL statement.*

- **Global buffers**: Global buffers are allocated memory resources regardless of the number of connections being handled. These buffers request their memory requirements during the startup process and retain this reservation of resources until the server process has ended.

When allocating memory to MySQL buffers we need to ensure that there is also enough RAM available for the operating system to perform its tasks and processes; in general it is a best practice to limit MySQL between 85 to 90 percent allocation of total system RAM. The memory utilization equations for each of the buffers is given as follows:

- **Per-thread Buffer memory utilization equation**:

  (read_buffer_size + read_rnd_buffer_size + sort_buffer_size + thread_stack + join_buffer_size + binlog_cache_size) * max_connections = total memory allocation for all connections, or **MySQL Thread Buffers** (**MTB**)

- **Global Buffer memory utilization equation**:

  innodb_buffer_pool_size + innodb_additional_mem_pool_size + innodb_log_buffer_size + key_buffer_size + query_cache_size = total memory used by **MySQL Global Buffers** (**MGB**)

- **Total memory allocation equation**:

  MTB + MGB = Total Memory Used by MySQL

If the total memory used by the combination of MTB and MGB is greater than 85 to 90 percent of the total system RAM then you may experience resource contention, a resource bottleneck, or in the worst case you will see memory pages swapping to on-disk resources (virtual memory) which results in performance degradation and, in some cases, process failure or connection timeouts. Therefore it is wise to check memory allocation via the equations mentioned previously before making changes to the memory buffers or increasing the value of `max_connections` to the database.

More information about how MySQL manages memory and threads can be read about in the following pages of the MySQL documentation:

- `http://dev.mysql.com/doc/refman/5.5/en/connection-threads.html`
- `http://dev.mysql.com/doc/refman/5.5/en/memory-use.html`

InnoDB memory and data file configuration

Before tuning any settings for the InnoDB engine, it's imperative to define the location for data files because MySQL uses these settings to initialize InnoDB during the installation process. Therefore, the variables cannot simply be changed after the files are created unless you follow specific data file migration processes. If one does not change the default settings then InnoDB data files will be initialized in the generic MySQL data directory as defined by the `datadir` variable.

Static variables

Static variables are only read during the MySQL process startup sequence. This means that once a static variable has been defined in the `my.cnf` file, it will be active during the entire runtime process and cannot be changed during operation of the database server process.

innodb_data_home_dir

This variable controls the location for shared tablespace files, status files, and InnoDB specific log files. If not explicitly set, this defaults to the value for the MySQL data directory. If you decide to enable the per-file tablespace setting then this variable will only determine the location of the shared data dictionary files while the per-file tablespaces files will be stored in schema specific directories. Professional opinion varies on this setting, as some administrators will want this to be set to the same explicit value as their general `datadir` value, and some will explicitly set this to a separate partition from the `datadir` value; each method has its benefits and, as such, the correct setting will depend on your environment's needs.

For example, `innodb_data_home_dir = /var/lib/mysql`.

innodb_data_file_path

This controls the specific InnoDB data file locations and sizes. Once set, this will be combined or concatenated with the setting specified in `innodb_data_home_dir`. The files specified must be larger than 10 MB and if supported by the operating system, they can be set to larger than 4 GB in size. This variable is a core element of the InnoDB capacity planning and performance scaling ability of the MySQL server. One of the more common settings is to create a baseline size file for the initial data directory contents, between 10 MB and 128 MB for the first file, and then a minimum size 10 MB file for the second file which is set to auto-extend status so that it can grow as needed. For example: `innodb_data_file_path = ibdata1:128M;ibdata2:10M:autoextend`.

innodb_log_group_home_dir

InnoDB redo log files are used to store transaction data before being committed and written to tablespace files on disk. These logs will be stored in the directory defined by this variable. Commonly this is set to the same location as the `innodb_data_home_dir` variable but for best performance, we recommend separating InnoDB data logging and data files onto separate physical storage arrays. This ensures that I/O resources do not conflict on servers that process high volumes of data or handle a high number of concurrent connections.

innodb_log_file_size

The size of your InnoDB log files can invariably cause or solve performance bottlenecks; as such it is important to chose the proper size for this variable. This defaults to a size of 5 MB, which is too small for most production workloads. Since the log files are initialized at their full size it is not possible to use the filesystem's reported file size to see if the logs are full, however one can inspect the file modification time to see how often the log files are rotated. The logs are written to in a round-robin manner, thus a rapid cycling of the log files, as indicated by close sequential modification time, will indicate that the logs are flushing too often and need to be increased in size.

Common settings for the `innodb_log_file_size` variable are typically seen as a function of the `innodb_buffer_pool_size` variable; on servers with relatively small buffer pool settings (under 4 GB) it is recommended to set the total size of the combined `innodb_log_file_size` variable to the same size as the buffer pool.

An additional requirement for maintaining performance of the log files is to ensure that the `innodb_log_buffer_size` variable is large enough to buffer changes in the RAM before being written to the log files themselves; the `innodb_log_buffer_size` variable is outlined later in the chapter.

As the buffer pool grows larger, it is not recommended to use a 1:1 ratio since the larger the logs the longer the crash recovery process will take if MySQL crashes, thus causing potentially extensive downtime. The maximum recommended setting for servers with high performance I/O for the log files, which is featured in the MySQL documentation, is to set the log file size to the value of the buffer pool divided by the number of InnoDB log files in use. For example, if one has a 10 GB buffer pool and four log files then the `innodb_log_file_size` variable would be set to 10/4 = 2.5 GB.

It is no coincidence that a server with high performance I/O will read and process InnoDB log file entries and finish the crash recovery process faster than a server with slower I/O, and as such it is wise to test the size of the log files prior to production use to determine approximate crash recovery timing values. The key difference between small log files and larger is that larger logs hold more commit data before requiring that data to be flushed to tablespace files. The log flushing and crash recovery process has improved over the course of MySQL releases so you may notice that later versions can handle larger log files without dramatic impacts to these situations.

Although this variable is static and can only be changed with a process restart, it also requires a specific process to change the value. Because of the way that InnoDB writes data changes to these logs files the size of the logs cannot simply be changed and then have MySQL start up cleanly. The logs must be either removed or moved and new log files of the requested size will be created. We will cover the process of resizing InnoDB log files in *Chapter 5*, *Maintenance and Monitoring*.

innodb_log_buffer_size

When InnoDB receives a transaction commit, it places the changes first to an in-memory buffer before writing to disk. This buffer, like many of the InnoDB memory buffers, helps keep data in fast access RAM so that the slower disks do not cause performance bottlenecks during high concurrency database traffic. As such, if we tune the `innodb_buffer_pool_size` variable to be larger than the default, we should also increase the size of the associated `innodb_log_buffer_size` and `innodb_log_file_size` variables. General recommendations for this variable range from 8 MB to 256 MB. The optimal setting will be determined by your workload and system resources.

Keep in mind that all changes must eventually be written to disk, so setting this buffer value very high will only buffer changes for a limited amount of time before eventually utilizing disk I/O resources. However, it can be used as a preventative measure to help with heavy I/O resource consumption.

innodb_log_files_in_group

This controls the number of log files that InnoDB will write to. The logs are written to in a sequential fashion. Unless there is a specific need, that has been tested to provide a performance or stability gain, this variable should be left to the default setting of `2`. If you need to resize the log files and provide more logging capacity, but do not want to engage the potentially hazardous process of removing and resizing the existing log files, then this value can be increased as desired and the InnoDB engine will simply create additional log files upon process restart. If you decide to decrease this variable and use fewer log files you must follow the same process used for resizing the InnoDB log files, which is outlined in *Chapter 5*, *Maintenance and Monitoring*.

innodb_buffer_pool_size

InnoDB is all about speed, and the biggest performance impact you can give an InnoDB based database is more RAM allocation to the `innodb_buffer_pool_size` variable.

InnoDB is an inherently disk based database engine, that stores and accesses database data that is stored on disk based filesystems. However, it is also very efficient in managing and utilizing Random Access Memory to buffer data during operation. One of the most important values that controls how InnoDB buffers data in RAM is the `innodb_buffer_pool` variable. The buffer pool is a memory-based cache for your table data as well as index data. This allows for rapid data access when compared to disk-based data caching. The log files, as we discussed are used to flush data changes from the buffer pool to temporary log buffers, also in RAM, to file based logs, and finally to disk. Before any of that flushing can occur, the data lives in the buffer pool so the larger the buffer pool is the more data it can hold, or buffer, between on-disk tablespace files and on-disk logs.

The buffer pool is managed by the **Least Recently Used** (**LRU**) algorithm with a mid-point insert strategy; this algorithm controls how data is added and purged from the pool by reading access timing values of data. The LRU method will remove the least recently used data which is stored at the end of the access list and then, due to the midpoint strategy, insert new data to the middle of the data access pool. The data will then be moved forward or backward on the access list depending on how often it is requested and used in relation to the rest of the data in the access list.

Before tuning this variable, check the version of MySQL in use; Version 5.0 uses a default setting of 8 MB which is unacceptable for most production cases, versions of 5.1 have differing values ranging from the previous standard of 8 MB on versions equal or less than 5.1.27 and a more realistic baseline setting of 128 MB on versions starting with 5.1.28 and greater.

When choosing a value for the buffer pool, common recommendations range from 80 percent of available system RAM to 100 percent of the size of the InnoDB data and indexes being used in your InnoDB based schemas. If, for example, your server has 64 GB of RAM and your total or working set of InnoDB data plus indexes are only 20 GB then it is likely that you will not see much benefit to setting the buffer pool at 80 percent of total RAM (64 * 0.8 = 51.2 GB). In situations like theses, it is more appropriate to allocate RAM resources elsewhere in the various MySQL memory buffers, such as the `innodb_log_buffer_size`, `join_buffer_size`, `read_buffer_size`, and others. To understand how MySQL allocates RAM to the various buffers you can refer the equations at the top of this chapter.

innodb_status_file

This variable offers the equivalent output of running `SHOW ENGINE INNODB STATUS` on the MySQL command line, but periodically written to a status file in the data directory. The output contained in the status file is useful for automated monitoring as it can be parsed for various bits of useful data that shows how InnoDB is operating. Additionally, it can be used during the troubleshooting process if there are operational issues with InnoDB. For timeline purposes the file is appended with the PID value.

transaction-isolation

This variable controls the method in which InnoDB handles the isolation of transactional data when multiple connections are present and accessing similar row content. As per the definitions listed in *Chapter 1*, *Getting Started with InnoDB*, the setting of the transaction isolation level will determine ACID compliance and support for XA transactions. Although this configuration setting is static at the server level, the isolation level can be set on a per-connection basis by your application if needed.

skip-innodb

If you ever need to disable the InnoDB engine this can be set in the `my.cnf` file. If you are having problems with InnoDB starting or not being available in the list of table engines, this would be a variable worth checking to ensure it is not enabled by mistake. In cases where InnoDB is not utilized for any tables, it is pertinent to enable this variable, thus disabling InnoDB so that MySQL does not allocate any resources to the InnoDB engine.

Summary

In this chapter we've covered the basics equations for determining memory allocation in MySQL as well as the basic InnoDB startup variables. We've also covered background information related to the LRU algorithm that InnoDB uses for buffering information in memory, and how it handles transactional data from clients and places information into RAM before flushing the data and writing it to disk.

In the next chapter, we'll dive into more advanced InnoDB settings and show how they affect database workload performance.

3

Advanced Configuration Parameters

So far we've covered the basic configuration parameters for InnoDB operation. The settings that follow will cover more advanced use and configurations as well as edge case operations. Additionally, these settings can aid in the process of troubleshooting issues with the InnoDB engine, fine-tuning the engine for specific needs, as well as providing an in-depth understanding of different operational states of transactional performance. It is of importance to note that these settings can drastically change how InnoDB handles our data and incorrect or flagrant changes to many of these variables, if used incorrectly, can result in data loss, data corruption, and server process failure. These advanced settings should only be used once the administrator has fully tested the database for stability.

InnoDB and input/output resources

As discussed earlier in the book, InnoDB's method for handling data change operation requires that data changes are stored in memory-based buffers and then written to log files and eventually to disk-based data tables. The process of writing to disk is resource intensive, from an I/O standpoint, and is the slowest part of the transaction commit process. As such, the performance of writing data changes to disk after a commit occurs is determined by:

- Speed of your disks
- RAID architecture and settings
- Underlying filesystem type and behavior
- The method that InnoDB uses to flush logs after a transaction is committed

InnoDB, as of MySQL 5.5, has variables to better handle high performance storage arrays so knowing the performance of your storage is a fundamental requirement in the tuning process.

We can test the performance of our I/O system with many tools. The most popular I/O specific test applications are as follows:

- SysBench
- Bonnie++
- Fio

A typical test aims to saturate the I/O system using repeatable testing methods so that the same test can be carried out on any number of machines, from which the statistical output of each machine can be compared for performance evaluation. The following commands are examples of general database related I/O testing methods:

SysBench

First we run a `prepare` command to set up the test files, and then we run the actual test with the second command. In this test we're running a sequential write test, as noted by `--file-test-mode=seqwr`, but we could also run the following test modes:

- `seqrewr`
- `seqrd`
- `rndrd`
- `rndwr`
- `rndrw`

For more information about SysBench please refer to their documentation available at `http://sourceforge.net/projects/sysbench/`.

Fio

The Fio testing program is very full-featured. The documentation and more examples can be seen on the developer's website at `http://freecode.com/projects/fio`.

Bonnie++

Long considered an industry standard for open source filesystem benchmarking, Bonnie++ has many in-depth tests available for fully customized load tests. Full documentation and sample tests can be referred to at the developer's page `http://www.coker.com.au/bonnie++/`.

Disk speed: standard IOPS

In general, we can refer to the following table to show the I/O capacity, also called **Input Output Per Second**, of current disks. Although a full discussion of I/O capacity planning is out of the scope of this reference manual, understanding the basics is a requirement for tuning the various InnoDB variables that relate to I/O usage. These values are approximates based on average disk speed results. Real world testing should always be done for capacity planning purposes.

Disk type	10K spindle speed	15K spindle speed
SATA	85-100 IOPS	120-130 IOPS
SCSI u320	150 IOPS	200 IOPS
SAS	150 IOPS	200 IOPS
SSD	2500 IOPS, NAND Flash, no spindles	

A very simplified example is, if our MySQL server is receiving 1000 committed transactions per second and each transaction must be written and flushed (default ACID requirements), that means our storage layer will need to handle 1000 IOPS, input output operations per second, or more if InnoDB has the double-write setting enabled.

In the most basic terms of IOPS capacity planning, in this example, if our server's data partition is hosted on a SSD drive then we can handle this load per the general IOPS performance numbers listed in the preceding table. However, if our data lives on spindle based disks then we will need to use some form of RAID, preferably RAID-10, to aggregate the disks and achieve higher IOP throughput to support our performance requirements.

Wherever possible, we can choose an RAID controller with a battery-backed cache, as it will offer both resiliency for maintaining the state during power outages and provide better performance than a controller lacking this feature. In general, high performance storage systems carry high dollar budgets, but in many cases we can tune InnoDB to achieve more transactions per second by changing flush plus write characteristics without changing our storage layer and thus we can operate with higher query traffic load without increasing expenditure on faster storage systems.

An important architectural consideration in regard to storage is that, if you are operating the MySQL data partition, regardless of the storage engine being used, on a RAID or SAN system that buffers changes through a volatile RAM-based caching system; changes are not preserved across system offline/online states then your database is at risk of data loss in power failure or hardware failure situations. If you use cache-based storage, be sure that it is non-volatile and maintains a battery-backed cache that ensures data in-cache is able to be processed and written to disk after a power failure scenario. Some hard drives, SSD included, offer firmware options for disabling volatile cache settings; we can use manufacturer's documentation as a reference to ensure that our data is not at risk.

If the cache is volatile then InnoDB may think it has preserved a write to disk only to have that data lost if the cache fails or is otherwise taken offline before the data is flushed from the cache and fully written to disk. If the storage provider allocates LUN devices, Fiber Channel or iSCSI, or otherwise to database servers, be sure to specify this requirement before operating in production.

InnoDB variables

InnoDB features two types of configuration variables:

- **Static**: Static variables only take effect on a process restart
- **Dynamic**: Dynamic variables can be changed at runtime via a `SET GLOBAL $key = $value;` query

Dynamic InnoDB variables

Dynamic variables are settings that can be set during process startup but then, if necessary, changed during runtime. This allows the database server to be tuned during operation without having to be restarted to change configuration parameters.

innodb_table_locks

This is a Boolean variable option that controls how InnoDB handles locking at the row level. InnoDB does not lock a full table by default in the way that MyISAM does, but sometimes a full table lock is necessary. This variable offers the ability for InnoDB to lock a whole table if using the `LOCK TABLES` SQL command, but only if your autocommit mode is set to `0`. Setting autocommit to a zero value, via `SET autocommit = 0;` during a query, will make MySQL commit changes to table data only after a `COMMIT` is executed, otherwise it will make changes immediately. This is specifically important because it affects the manner in which transactions are handled and how our data is protected against consistency and ACID compliance.

This should only be changed from its default setting of `1` if you have a specific need to alter the method in which transactional data is committed to the database. During troubleshooting procedures, this is a good variable to check to ensure that the database is operating as expected. This variable has had minor changes in edge case behavior for `LOCK TABLES…WRITE` commands after MySQL Version 5.5.3; one should check the version to determine if it would affect the server.

innodb_flush_log_at_trx_commit

Next to `innodb_buffer_pool_size`, the `innodb_flush_log_at_trx_commit` variable is one of the most effective variables for increasing transaction commit performance. By default this variable is set to a value of `1`, which means that InnoDB will write the contents of the memory-based log buffer to disk-based log files and issue a flush on the log file after each transaction commit. Depending on the performance characteristics and I/O capacity of your server's filesystem, this flushing operation can cause extensive I/O usage to the point of creating a performance bottleneck that may block other database operations. We can change how often InnoDB flushes write to the disk subsystem by changing this variable. Alternate values for this variable include `0` and `2`.

A value of `0` will cause InnoDB to write the contents of the log buffer to log file once per second along with a flush execution on the log file at the filesystem level; this is done regardless of the number of commits per second.

A value of `2` will cause InnoDB to write the log buffer to disk during each commit but not flush on each commit. Additionally, depending on the I/O scheduler in use on the filesystem, InnoDB may or may not flush the log file to disk once per second although it will attempt to do so. Testing and choosing the proper I/O scheduler for your filesystem is critical if setting this variable to a value of `2` to ensure that InnoDB's I/O events are handled as expected.

There are benefits and drawbacks to changing this variable. Performance gains are not free; they are typically a compromise among speed, stability, and cost. In the case of performance gained by changing the write and flush characteristics of the InnoDB storage engine we are compromising stability and data integrity of the transaction processing sequence. The default setting of `1` guarantees the ability to provide ACID compliance, any other setting will disable ACID compliance; in particular this affects the durability characteristic. If `innodb_flush_log_at_trx_commit` is set to `2` or `0` then we risk losing up to one second of transactional data that the application thinks it has committed in a durable manner. Specifically, when set to `0` we can lose data if the MySQL process crashes, while a setting of `2` will only cause data loss if the physical server itself suffers a critical failure such as power loss. So, in order of data integrity versus performance gain, the safest and least performance oriented setting is the default of `1`, then `2`, and then `0`.

innodb_io_capacity

As we covered at the start of the chapter, InnoDB relies on RAM before it writes to disk, but eventually every bit of changed data has to make it to disk in order to remain durable. New in MySQL 5.1, InnoDB is able to be fine-tuned for higher performance storage systems by setting an upper limit on storage I/O capacity. The default setting for this value is 200 IOPS as of MySQL 5.5, prior versions hardcoded this to 100. The default setting is equivalent to running the storage layer on a single `15k` SAS drive (or RAID-1 array using the same disks), or a four disk RAID-10 SATA drive array; these are common local disk storage setups for MySQL servers. However, if you are using a larger disk array or a fiber attached DAS or SAN then this variable can be tuned much higher to support the increased IOPS performance. When tuning this variable it is wise to also tune the following variables:

- `innodb_write_io_threads`
- `innodb_read_io_threads`

innodb_thread_concurrency

The default value for this variable is set to `8`, but on high performance servers this value can cause InnoDB performance latency due to the method used to handle thread execution. When InnoDB receives workload requests, it requests a work thread from the operating system and sets about to execute the workload. The maximum number of concurrent work threads that InnoDB will use at any given time is limited by this variable, thus by default InnoDB will only use up to eight work threads to execute workloads. If InnoDB needs additional worker threads once the maximum number have been allocated, they will be placed in a FIFO queue to be allocated and processed once pending completion or termination of the existing work threads. This method of waiting for a thread lock can cause queue saturation and delays to transaction processing.

Servers with a high number of threads will generally benefit from having this variable increased to the order of 50 to 100 percent of the total execution threads. If your server supports Intel's HyperThreading and is enabled, it is best to not consider these virtual threads for use in the tuning process and instead divide the total reported threads by a factor of two to get the total physical threads.

Depending on the version of MySQL being used, this variable is interpreted differently. Prior to 5.1.12, a setting greater than `20` is considered infinite and will be treated just the same as 21 or 221, which potentially causes confusion during the tuning process. However, later versions do not suffer this infinite status. On versions after 5.1.12, this variable can be set to a value of `0` to disable concurrency checking entirely, which allows InnoDB to use as many threads as the operating system will allocate it; use caution with this setting especially if you are running other server applications on the same server, such as an integrated LAMP server, as InnoDB could potentially use all possible threads and starve other system processes

innodb_commit_concurrency

InnoDB, by default, will use as many threads as it can allocate to commit data. This can be problematic in some cases, especially when using virtualization without resource quotas or in cases of multitenancy and integrated services platforms with shared resources. As such, if we are profiling InnoDB to only use a specific set of total available resources, this can be tuned to limit the number of concurrent commit threads.

Although this is a dynamic variable it cannot be changed, as of 5.1.36, to/from a value of zero during runtime; however dynamic control of values not including zero is permissible.

innodb_support_xa

Enabled by default, this can be changed to prevent InnoDB from supporting two-phase global transaction commit queries. Additionally, if your server is running more than one thread and utilizes the binary log system for replication or other archival / point-in-time recovery, this setting is essential for ensuring that data changes are written to the bin logs in sequence. Having data recorded in order is of critical importance in replication scenarios, point-in-time recovery, and all ACID systems.

The only times this setting should be disabled is in the case of a master server that writes changes only via a single thread, thus being unable to concurrently execute changes to data out of order, and on single-threaded replication servers where replication inherently runs via a single execution thread. Disabling this setting in those scenarios can increase performance but care should be taken not to copy this setting from one server to another without taking those use cases into account, otherwise data corruption and/or replication failures can occur.

innodb_fast_shutdown

By default InnoDB will issue a fast shutdown when a service stop is issued. This means it skips full-purge operations and does not issue an insert-buffer merge procedure. On a large database with heavy insert workloads this saves time during the shutdown process, which is ideal for most scenarios.

A MySQL crash is the equivalent of setting this variable to a value of `2` and is usually reserved for use cases that require an immediate halt to the database. In this event, InnoDB will issue and attempt a crash recovery during the next process startup. Crash recovery can take an extensive amount of time depending on the active workload at the time of the crash.

Appropriate times to set this variable to a value of `0`, which issues a full purge and insert buffer merge, applies to scenarios such as:

- Prior to a MySQL upgrade or downgrade
- Shutting down the database for a cold binary-data migration
- Physical relocation of a server
- Adding, growing, or shrinking InnoDB log files

Static InnoDB variables

Static variables are only read during the MySQL process startup sequence. This means that once a static variable has been defined in the `my.cnf` file it will be active during the entire runtime process and cannot be changed during operation of the database server process.

innodb_file_per_table

There are several typical methods used when allocating and managing on-disk database tables:

- Raw partition tablespaces
- Shared file tablespaces
- Single file per table

As usual, when solving problems, there are multiple solutions where each offers benefits and detriments, thus some are better suited to certain use cases than others. The variable for `innodb_file_per_table` is one such architectural choice for managing data that fits very well for its intended use case; separating data out into uniquely named groups, and managed at the resource level. Quite simply, this variable configures InnoDB such that it will create and store table data and associated indexes in a private, non-shared, single data file for each InnoDB table. Later in the book we'll cover use cases where this variable does not apply or is not suited.

The following benefits are attributed to use of this data management method:

- Easier to manage data files if using symlinks to distribute large tables to different partitions
- Easier to troubleshoot I/O resource usage on a per-table basis as files can be watched to see which is using more/less I/O traffic compared to watching shared tablespace files
- Easier to repair and recover from corrupt data:
    - Data is separated per table, so one corrupt table does not affect the rest of our InnoDB tables
    - Exporting and re-importing data for a table only affects the single tablespace file instead of larger aggregate shared tablespace
- Allows the DBA to distribute tables to separate files to work around filesystems that operate with a single file size limit (for example, 2 TB in Ext3) when dealing with very large schemas
- If operating a large number of deletes from a table it is far easier to reclaim filesystem space on disk with `innodb_file_per_table` than a shared tablespace due to the way InnoDB purges data and reserves blocks on disk
- Less fragmentation compared to shared tablespaces and easier to defragment (`OPTIMIZE TABLE`)
- If running MySQL in a multitenant situation, shared hosting for example, then each client/schema will be in separate files instead of in a shared tablespace, this can be seen as a security benefit in some audits
- Easier backup processes
    - Cold copy binary data based backups across the wire will send and buffer smaller individual tablespace files and index files faster compared to one potentially very large shared tablespace, that cannot be split into smaller files without complex chunking and checksum processing and verification. Additionally, we cannot issue multiple copy threads to utilize more network bandwidth since the single tablespace file can only be transmitted via a single stream.

- Hot backup copies only existing data, not pre-allocated empty data, if using InnoDB Hot backup on a shared tablespace that is pre-allocated with 100 GB but only 25 GB of that space is in use with active data, then the backup will still copy all pre-allocated 100 GB for the backup. If we're using `innodb_file_per_table` and our total combined tablespace files have 25 GB of data on disk then InnoDB Hot backup will generate a 25 GB backup set.

As you can see there are many benefits to enabling this variable. One must choose wisely, and be sure to read the section later in the book that covers alternate architectures to see the benefits of shared and pre-allocated tablespaces.

The tablespace files will live in their respective schema directory instead of nested inside the shared tablespace files that are created by the variable definition: `innodb_data_file_path`. If you set this variable after InnoDB tables have been created, the data will not be migrated to unique files but instead will continue to be stored in the shared tablespace files as previously used. Only new tables created after enabling this variable will be separated into their own tablespace files. To functionally shrink or remove content from the shared tablespaces the data must be exported out of the tablespace, via backup to SQL or other flat format, and then the tablespace can be removed and data re-imported.

innodb_additional_mem_pool_size

This variable controls the amount of memory allocated to the InnoDB data dictionary, which contains essential reference and metadata about the contents to the InnoDB data being stored. The default setting is 8 MB, so if you have many tables this is a good variable to increase. Typical settings include reasonable amounts of RAM from 16 MB to 128 MB. InnoDB will only store as much data here as it needs, so increasing this above the needed size will not speed up transaction processing. MySQL has handled this buffer setting in different ways over the years so referring to the documentation for your version of MySQL is recommended prior to making any adjustments.

innodb_buffer_pool_instances

New as of MySQL 5.5, this variable allows enhanced control over memory management by distributing the configured value of the `innodb_buffer_pool_size` variable into *N* separate regions. This only takes effect if the buffer pool is sized 1 GB or larger. The main benefit of changing this from the default value is to increase concurrency when using larger buffer pools that have a high rate of data being changed. MySQL recommends setting this to a value such that each buffer pool instance remains at a minimum size of 1 GB or more. The performance improvement comes from a reduction of access contention in cases where there is a large amount of read and write traffic on the buffer pool.

innodb_write_io_threads

This controls the number of background I/O threads that InnoDB will use for various write based workloads. The default setting is `4` but it can be set as high as `64` on high end storage systems that support higher I/O operations per second. Keep in mind that each background thread can handle up to 256 pending operations, so this variable is a multiple of that number. This can be a difficult setting to predict optimal settings for but the sane process is to watch for `>64 innodb write io` requests in the InnoDB status output, which would generally indicate that this variable should be increased in order to lower the pending requests. For more information on how to tune InnoDB for I/O performance you can refer to this section of the MySQL documentation: `http://dev.mysql.com/doc/refman/5.5/en/optimizing-innodb-diskio.html`.

innodb_read_io_threads

This variable is similar to `innodb_write_io_threads` but specifies the number of read I/O threads that InnoDB can utilize. Tuning for this value is done in a similar manner to the write threads variable.

skip-innodb-doublewrite

InnoDB enables the `innodb_doublewrite` setting by default, and as such it will write all incoming data two times. Data will be written to the doublewrite buffer and then to physical data files to ensure data integrity in the event of power loss or other system failure. This can be disabled via the `--skip-innodb-doublewrite` configuration setting to achieve maximum write performance as it reduces the required write workload on the I/O system.

Summary

In this chapter we've covered advanced use and tuning of InnoDB startup variables as well as discussed the basics of InnoDB IOPS management and I/O architecture as it relates to MySQL servers. In the next chapter we will cover the methods used to load test and monitor InnoDB for performance.

4
Load Testing InnoDB for Performance

Load testing InnoDB for performance requires more than simple observation of the database engine itself. In order to understand how InnoDB is performing, we need a full view of the system health and status. During formal load tests we will collect metrics about InnoDB concurrently with metrics from the operating system, storage system, CPU, and memory.

In this chapter we will review common tools used for:

- Linux- and Unix-based operating system performance monitoring
- Storage system benchmarking
- Various InnoDB flush methods
- Open source based load testing applications

Prior to discussing specific methods used for load testing, InnoDB databases we will go over the general philosophy of load testing and procedures used to determine progress. Without properly understanding the methodology behind load testing, our tests are effectively a group of useless numbers where correlation between results is impossible. The main purpose of load testing is to generate metrics and analyze results in a repeatable fashion through which we can prove or disprove a hypothesis about a system under load.

Load testing allows us to see how a defined system functions under a defined load. That load can be in many different forms:

- Static or dynamic quantity of connections
- Queries per second
- Dynamically growing data set size

- Quantity of concurrently executing threads
- Many other methods of generating and sustaining load

The main goal in our tests is to be able to predict, with a healthy margin of error, how a system will respond to defined factors so that we can reliably predict resource utilization and generate capacity plans for production use. You can read more about the load testing methodology here at `https://en.wikipedia.org/wiki/Load_testing`.

Load testing also helps prevent unexpected surprises or detrimental impact to performance when making changes to configuration settings, or base operating system variables. Load testing prior to production alteration allows us to observe, collect, and analyze performance metrics, thus enabling us to set expectation with clients, system users, or adherence to a defined Service Level Agreement.

The load testing method is an extension of the scientific method that is tailored to systems and database engineering. We operate via the following steps:

1. Question.
2. Definition.
3. Test and observe.
4. Analyze and hypothesis.
5. Prove/disprove via test.
6. Publish results.

Here is a sample load test scenario to explain a common load testing hypothesis:

1. Generate a question about which we can test objectively. For example, under heavy traffic, if we increase the InnoDB buffer pool size will that take the load off of the I/O system and allow for higher **Queries Per Second** (**QPS**) performance?
2. Define our environment and explain our test. For example, 8-core xeon server with RAID-10 15 k-SAS drives, executing 1000 connections per second with complex transactional queries for one hour of time, `innodb_buffer_pool=X`, and so on.
3. Execute the test and collect metrics that will be used in the later steps. For example, execute SysBench load test on our 8-core xeon server with 1000 connections per second for one hour of complex transaction queries with the `innodb_buffer_pool=X`, and so on.

4. Analyze our metrics and form a hypothesis about the test results. For example, our queries per second were equivalent to `N` prior to changing the buffer size variable, the system resource utilization metrics show that our performance bottleneck suffered from high I/O usage while RAM resources were not fully utilized; therefore the hypothesis is to increase the InnoDB buffer pool memory variable to utilize more RAM, this will allow more data to be stored in RAM, which will take load off of the I/O system thus providing an increase to our queries per second performance.
5. Using reproducible test methods such as a test profile in Quadrant or OLTPbenchmark, we execute a new load test and collect system and database resource utilization metrics. We then analyze the test metrics results in regard to our hypothesis and come to a conclusion about whether our hypothesis was correct or incorrect. Wherever applicable, generate additional iterative load tests as needed for analysis and fine-tuning. In our example, increasing the size of the buffer pool allowed InnoDB to move data from the I/O subsystem into RAM which allowed for faster data access, which resulted in a higher number of executed queries per second. Our hypothesis is now proved, we move on to the final step.
6. Wherever possible and if results are deemed useful, we publish our results to the professional community for review. Some load tests, depending on your job and data being used, are bound by Non-Disclosure Agreements, which limit the ability to publish results to the public.

The previous scenario is a very common issue that comes up in the MySQLtuning process. Using a formal load test allows us a unique view into the operational state of the database during simulated use and, when executed correctly, load test results can help us calculate iterative performance gains to solve scaling challenges.

Open source system monitoring tools

Before we run any load tests we need to be able to collect system load metrics. These will be used to test the hypothesis and to monitor activity during the testing periods. Testing without usage statistic monitoring is equivalent to blind operation that prohibits objective reasoning. Useful tools for system resource monitoring are included with Linux and most BSD operating systems:

- **sar**: This allows the administrator to collect, report, and save system activity information. This can be used in an interval collection state with many advanced options for resource monitoring. A number of supporting tools are available that generate graphs from sar output. Sar can be found in the `sysstat` package on most Linux systems.

- **top**: This displays current active operating system tasks. If run without flags, it will only report tasks owned by the active user. Referencing the manual page will describe the numerous flags available to alter information that is displayed.
- **iostat**: Part of the `sysstat` package of monitoring tools, this is a requirement for monitoring I/O resource activity during load testing. If this is not installed by default, it is highly recommended that you install and use it during all load testing.
- **free**: This shows current memory usage and can be parsed during a simple bash loop. A more effective way to get this information is direct via the proc entry.
- **ps**: Process monitor and snapshot reporting. This provides many usage statistics including: cpu, memory, virtual memory, swap, pid, command, process owner, threads, and many more.
- **/proc**: This directory holds most of the available system statistics. Useful items for load testing include, but are not limited to the following:
    - `/proc/loadavg`
    - `/proc/meminfo`
    - `/proc/partitions`
    - `/proc/diskstats`
    - `/proc/cpuinfo`
    - `/proc/swaps`
    - `/proc/vmstat`

Open source MySQL load testing applications

Several industry standard tools exist for load testing MySQL, and by extension any tables you are using the InnoDB engine on. We will focus on the open source options mentioned in the following sections:

Log replay

Before discussing specific applications it is important to understand that if we have an existing production server, that is executing and recording traffic via the binary logs, then we have available data for use in load testing without having to rely on other applications to generate sample test data. This is called **log replay** and the method is simple:

1. Use recorded transactions from MySQL binary logs.
2. Convert the binlogs to SQL files using the `mysqlbinlog` command.
3. Execute the SQL files against the database server(s) in question.

This directly executes the same queries that were executed on the production server. It is of vital importance to note that if you are getting logs from production, you should not execute them again on the production servers as this can cause all number of issues. Log replay should only be used in staging, development, or dedicated load test environments.

While log replay is not a substitute for a full load test, the SQL queries can be used in conjunction with one of the tools discussed in the following sections to test performance differences while using repeatable and consistent real-life application traffic. These SQL queries are specifically useful when troubleshooting performance issues from specific time/date periods since we can replay the queries and test iterative changes to our configuration to, hopefully, address performance concerns.

SysBench

SysBench (`http://sysbench.sourceforge.net`) is one of the most commonly referenced MySQL benchmarking applications and for good reasons, as it supports the majority of general load testing tasks in a simple to use command line driven script. By default it will create an InnoDB-based table if you are running the transaction tests, but it is wise to check the table engine prior to starting your tests. SysBench by itself does not offer options for iterations, dynamic sequences, or graph output; these features need to be provided via a wrapper application or other method. The included tests are as follows:

- CPU intensive
- File I/O tests
- RAM tests
- Mutex tests
- Transactional query tests (complex, simple)
- Non-Transactional tests, read-only query testing
- Options for multi-threading
- Options for multi-node cluster aggregate testing

Quadrant framework

The Quadrant framework (`http://code.google.com/p/quadrant-framework`) is a project that combines SysBench, DyGraphs, and Python into a load testing framework to help create and automate repeatable load tests while offering visual metric analysis via automatic graph generation. Quadrant offers the following features:

- Standardize your SysBench tests without requiring you to write one-off bash scripts to handle looping and iteration commands
- Graphing output via Dygraphs
- CSV test result output for easy import to data visualization systems or import into an SQL database
- Customize your iterations via ability to watch an unlimited number of MySQL global status variables
- Customize your iterations via ability to alter MySQL global variables between iterations
- Run arbitrary SQL commands between each iteration
- Run arbitrary system commands between iterations
- Allows you to control the caches between iterations
- Set custom default values and create configuration templates for easy repeatability and portability of your tests
- Extensible code base which is written purely in Python 2.6

OLTPbenchmark

OLTPbenchmark (`http://oltpbenchmark.com`) is a load testing application specifically tailored to OLTP databases. It offers a productive mix of repeatability and modular testing, with the following features:

- **Precise rate control**: This allows to define and change over time the rate at which requests are submitted
- **Precise transactional mixture control**: This allows to define and change over time percentage of each transaction type
- **Access Distribution control**: This allows to emulate evolving hot-spots, temporal skew, and so on
- **Support trace-based execution**: This is ideal when it comes to handling real data

- **Extensible design**
- **Support for statistics collection**: This takes care of microseconds latency and throughput precision, seconds precision for OS resource utilization
- **Automatic rendering via JavaScript**: Even facilitates rendering via many other available and gnuplot matlab scripts
- **Elegant management of SQL Dialect translations**: This targets various DBMSs
- Targeting all major relational DBMSs and DBaaS offerings via the standard JDBC interface (tested on MySQL, Postgres, Oracle, SQLServer, DB2, HSQLDB, Amazon RDS MySQL, Amazon RDS Oracle, SQL Azure)
- **Store-Procedure friendly architecture**

MySQL Benchmark Suite

MySQL Benchmark Suite is a toolkit for basic load testing (`http://dev.mysql.com/doc/refman/5.5/en/mysql-benchmarks.html`). If you install MySQL from source, or download the `mysql-bench` package, you can use several very basic testing scripts. These have limited features, including being limited to a single execution thread, and should only be considered if there are no other means of load testing available. Consult the MySQL documentation for more information.

MySQLslap

MySQLslap (`http://dev.mysql.com/doc/refman/5.5/en/mysqlslap.html`) is commonly used, but simple script is included as a client application with the GA distribution of MySQL. It is typically used to generate load or function as a simulation mechanism in a larger scale custom load testing system where it is wrapped via a parent script or application to control data visualization and iteration controls. It has the following options:

- Iteration pre-test command execution + sql query
- Iteration post-test command execution + sql query
- Per-connection query limits
- Connection concurrency testing
- Bulk query execution
- Auto-generated query execution
- Index test controls (limits, quantity, and so on)
- Per-thread read/write controls

Filesystem benchmarking

When we work with MySQL we are inevitably interacting with the filesystem in more ways than just storing data. We're requiring the filesystem to:

- Operate on blocks of data as fast as possible to read and write log files
- Store and process temporary tables for sorting larger tables that cannot be processed in-memory
- Also provide redundancy in the case of hardware failure

If we can run our entire database in-memory then it will almost always be quick to respond, but for databases that are larger than the available system RAM, we need to tune the filesystem to perform at its best. In order to tune the filesystem and ensure optimal settings for our workload, we will apply the same testing methodology that was explained at the start of the chapter, except we will focus on the I/O resources. When running filesystem benchmarks be sure to run more than one, and preferably at least ten iterations so that a realistic average can be seen with outliers removed. To do this we can use the following common tools to define our available IOPS and then measure IOPS utilization:

hdparm

Included with the GNU/Linux operating system, this tool reports on current hard drive statistics and configurations. It also provides an interface to configure the hardware to maximize or operate in conjunction with specific database configurations.

Of particular interest are the `hdparm` command options:

- `--direct`: This sets the device to operate via the `O_DIRECT` flush method, which will be covered later in the chapter. From the manual page:

  > *This bypasses the page cache, causing the reads to go directly from the drive into hdparm's buffers, using so-called "raw" I/O.*

- `-g`: This displays drive geometry.
- `-t`: This performs device read timings.
- `-T`: This performs cache read timings.
- `-i`: This display drive identification.
- `-I`: This provides detailed/current information directly from drive.
- `-w`: This disables write cache, which can be of particular use on RAID and SSD based volumes.

Bonnie++

An I/O specific benchmark tool, this will report on the performance of our storage system, which we can later use to tune InnoDB settings for I/O configuration. The following is a sample command which we can reference during our InnoDB tuning process:

```
» bonnie++ -r 512 -s 1024 -d /tmp -f -b -n 1
```

The command flags define the following:

- `-r`: This specifies the RAM to use; can be explicitly set or Bonnie++ can set based on the machine's reported RAM specifications
- `-s`: This helps to synchronize before each test
- `-d`: This defines the directory to write files to during the test
- `-f`: This is a fast mode control; skips per-char IO tests if no parameter, otherwise specifies the size of the tests for per-char IO tests (default 20 M)
- `-b`: This specifies no write buffering; issues a `fsync()` after each write
- `-n`: This specifies the quantity of tests to run

Fio

This is another I/O specific benchmarking tool, that can be used to create batch load testing jobs based on configuration files. The benefit of this application over Bonnie++ is arguably more detailed and easier to interpret reports.

InnoDB Flush methods

InnoDB handles disk flushing in different ways, interfacing with the filesystem to ensure data is written to disk. The following options apply and can affect the write performance in dramatic ways. These settings should be benchmarked on your system before production use to ensure you're getting the most out of the hardware capabilities. In some instances a simple change of these settings can double the number of queries per second that are able to be executed due to write performance efficiency.

- **fdatasync** (the default setting): InnoDB uses the `fsync()` call to flush data and logs to disk. Good for most workloads but remains a conservative choice and is prone to I/O performance penalty.
- **O_DSYNC**: This can provide a benefit for `SELECT` intensive workloads where InnoDB files are located on a fiber optic attached DAS or SAN device.

- **O_DIRECT**: This can help prevent double-buffering of data between InnoDB and the filesystem caches as files can be read via raw I/O.

InnoDB Flush method comparison chart

Setting	**Open data files**	**Flush data file**	**Open+Flush log files**
`fdatasync`	`fsync()`	`fsync()`	`fsync()`
`O_DSYNC`	`fsync()`	`fsync()`	`O_SYNC()`
`O_DIRECT`	`O_DIRECT()`	`fsync()`	`fsync()`

Thread concurrency testing

Threads, or number of concurrent connections to the database, should be tested in an iterative manner. When testing this operational capacity limit, which will be set via the `max_connections` startup variable, it is imperative to not change the test value dramatically over a short period. A conservative value for iterations would be a value no higher than a `50` connection increase per iteration for servers with large amounts of RAM available for the per-thread buffers, or no higher than a `10` connection increase per iteration on servers with smaller amounts of available RAM.

Advisory on transaction-isolation level

If your application does not require full ACID compliance, the transaction-isolation setting can be experimented with via load testing. If testing settings that are not `READ-COMMITTED` or `SERIALIZABLE`, be sure to disable any reliance on transactional data integrity in your application or you may suffer issues with referential integrity and other data consistency issues.

Summary

In this chapter we've covered the basics of load testing methodology, open source tools available for system resource metrics, as well as the typical applications used for load testing I/O systems and MySQL. After absorbing this content you should be able to objectively prove performance gains or losses when tuning performance settings on both the operating system level as well as the MySQL database application. In the next chapter, we will cover typical maintenance operations as well as common monitoring techniques for InnoDB.

5
Maintenance and Monitoring

Monitoring and reporting on InnoDB usage is a key element of a healthy and stable database operations environment. Using statistical information about the health and stability of InnoDB, we can use preventative maintenance procedures to ensure optimal performance as well as prevent potential downtime.

Maintenance procedures

The InnoDB engine should be monitored on a regular basis to ensure consistency of resource usage and overall health. Common maintenance procedures involve increasing log file sizes, tuning buffers for changes in application workload, adding tablespace files, and the list goes on. Here are some of the most common maintenance tasks explained via procedure.

Adding and resizing InnoDB log files

InnoDB log files, especially if left at their default settings, sometimes need to be increased in size or in quantity. This is useful if we need to increase the amount of transactional logging capacity or spread the writes to additional log files. This process requires downtime for the server and MySQL must be stopped when the log files are worked with, otherwise data loss will occur.

The process is as follows:

1. Stop the MySQL service, then ensure it is not running.
2. Change directory to the location specified in `innodb_log_group_home_dir`:

   ```
   $> cd /var/lib/mysql
   ```

3. Move files to the `tmp` location for back-out purposes:

   ```
   $> sudo mv ib_logfile* /tmp/
   ```

4. Change the quantity or size of log files by editing the system's `my.cnf` file. To increase the size, change the value of `innodb_log_file_size`. To change the quantity of log files, increase the value of `innodb_log_files_in_group`.
5. Start MySQL and check to see if the new files have been created in the log file directory. The new log files will be automatically created with the new settings by InnoDB when it finds them missing from the directory:

```
$> sudo /etc/init.d/mysqld start
$> ls -al /var/lib/mysql/ib_log*
```

6. Monitor the MySQL error file. At this point the process will complete.

Adding tablespaces to InnoDB

As we covered in previous chapters, InnoDB has multiple methods for storing data in tablespace files. If you are not using `innodb_file_per_table` then InnoDB will store table content in shared tablespaces. If you have not enabled the `autoextend` setting on your tablespaces, defined in `innodb_data_file_path`, then you may find your tablespaces running out of room to store data as your database grows. In these scenarios you will need to extend the capacity by adding additional tablespaces. The process is as follows:

1. Stop MySQL and create a backup of the configuration file for back-out purposes.
2. Edit the system's `my.cnf` file and change the value for `innodb_data_file_path` while adhering to the following requirements:
    - The variable is defined using the following string: `ibdata1:$size;ibdataN:$size;…ibdataN:$size;`. The correct values for the size variable will depend on your environment.

      The value of `$size` is a value in regular bytes or K for kilobytes, M for megabytes, and G for gigabytes. If you are adding a tablespace where one was previously defined without the K,M,G shorthand or if it was set to `autoextend` then you will need to define the tablespace size in exact byte value or InnoDB will refuse to start, stating that it has encountered a tablespace file of incorrect size.

      The value of `N` is the sequence value for the number of tablespace files being used. InnoDB's default setting is to create a single 10 MB `autoextend` file. If this is the case, and your `my.cnf` file does not have an entry for `innodb_data_file_path` then you will define the first tablespace by its current byte size and add additional sequence files as needed.

3. Currently defined tablespaces or default tablespaces cannot be changed, but additional tablespaces can be added to InnoDB by incrementing the file sequence via additional `ibdataN:$size` values to create new tablespace files.
4. Start MySQL and watch the error log for any issues.

Utilizing RAW tablespaces

InnoDB has an optional method of storing tablespace data in RAW partitions. This can increase I/O performance in some cases because it removes the abstraction and management layer that an operating system's filesystem tacks onto all database read + write operations, and on applicable storage devices and also allows InnoDB to perform non-blocking writes. This method of storage is equivalent to "removing the middle man" and allowing InnoDB to write directly to block storage without interacting with a logical filesystem. The advantages of this storage method should be tested on systems before use; not all systems will show the same performance benefits.

To enable RAW tablespaces we follow this process, which assumes you are starting with non-formatted partitions that do not contain any data or filesystems.

1. Stop MySQL and create a backup in the configuration file for back-out purposes.
2. Define your partitions in the `innodb_data_file_path`. In this example we have two partitions to use (`/dev/sdc1` and `/dev/sdd1`), each of size 10 GB.

   ```
   innodb_data_file_path=/dev/sdc1:10Gnewraw;/dev/sdd1:10Gnewraw
   ```

3. Start MySQL and monitor the error log.
4. Once InnoDB finishes initializing the partitions we will shut down MySQL and change the `newraw` definition to simply `raw`. InnoDB uses the `newraw` definition to flag the partition as requiring initialization, so once it has been initialized it needs to be flagged as available for storage use.
5. Stop the MySQL server process.
6. Edit the `my.cnf` and change `newraw` to `raw` on the `innodb_data_file_path` variable.
7. Start MySQL, then log in and run the following command to ensure the partitions are listed as expected:

   ```
   mysql> SHOW GLOBAL VARIABLES LIKE 'innodb_data_file_path';
   ```

Monitoring InnoDB

Timely monitoring of InnoDB is an essential aspect of the health and happiness of the database engine. Without consistent monitoring in place we turn a blind eye to the database and hope things go well during use; this cannot be done in a production environment so we need to know how to properly retrieve information from InnoDB so that we can see usage trends and predict and prevent issues before they occur.

Monitoring via show table status

There are several useful pieces of information that can be obtained from the `show table status;` query. In regard to InnoDB based tables we can use this to find the following information. Keep in mind that some of this information can be monitored via other, more automated means:

```
Version: <version of the table's format file>
Row_format: <Compact / Redundant / Dynamic / Compressed>
Rows: <approximate number of rows. Use "select count(*) from <table>;"
to find the exact count value>
Avg_row_length: <number of avgerage rows>
Data_length: <length of data file>
Max_data_length: <max length of data able to store in the data file,
given the current pointer being utilized. >
Index_length: <length of index file>
Data_free: <MySQL >= 5.1.24, this shows the available bytes in the
related table-space file>
Auto_increment: <incremental value for auto-incrementing row>
Comment: < prior to MySQL 5.1.24 this shows the available bytes in the
table-space file>
```

Querying INFORMATION_SCHEMA

We can get a wide array of information about InnoDB from the `INFORMATION_SCHEMA` database, which stores internal data for the MySQL server during operation. The following tables are available for InnoDB information:

- **INNODB_CMP**: This table contains information for compressed data format operations. Reference for column details and definitions: `http://dev.mysql.com/doc/innodb-plugin/1.0/en/innodb-information-schema-innodb_cmp.html`.

- **INNODB_CMP_RESET**: This table contains the same information as `INNODB_CMP` but functions as a counter-based metrics table whose data is truncated after each table read. For example, if you select from this table every five minutes then you will get metrics from the previous five minute period versus selecting data from the `INNODB_CMP` table which returns aggregate metrics since the start of the InnoDB process. **INNODB_TRX**: This table contains detailed information about current transactions that InnoDB is executing. This is one of the most useful query profiling tables available for InnoDB monitoring purposes and transaction trending. The columns include the following:
    - **TRX_ID**: This contains the transaction ID
    - **TRX_WEIGHT**: This contains transactional weight value as determined by InnoDB, based on other current transactions
    - **TRX_STATE**: This contains transaction operational state such as running, lock wait, rolling back, and committing
    - **TRX_STARTED**: This contains the timestamp value for start time of transaction
    - **TRX_REQUESTED_LOCK_ID**: This contains the ID value of the transaction being waited on if currently in lock state. We can query the `INNODB_LOCKS` table to find further information about the lock we're waiting on via this query: `SELECT a.*, b.* FROM INNODB_LOCKS AS a, INNODB_TRX AS b WHERE b.TRX_REQUESTED_LOCK_ID = a.LOCK_ID;`
    - **TRX_WAIT_STARTED**: This contains the timestamp of lock wait mode initiation
    - **TRX_MYSQL_THREAD_ID**: This contains the allocated thread ID
    - **TRX_QUERY**: This contains the query being executed
    - **INNODB_CMPMEM**: This table contains the information for compressed page data that is stored in the buffer pool, including: page size, pages used, pages free, relocation operations, and relocation timing.
    - **INNODB_CMPMEM_RESET**: This contains the same information as `INNODB_CMPMEM` but functions as an interval-based table, where all data points are reset to zero after being selected.
    - **INNODB_LOCKS**: This contains information pertaining to existing locks. Full column details are available here: `http://dev.mysql.com/doc/innodb-plugin/1.0/en/innodb-information-schema-innodb_locks.html`.

- **INNODB_LOCK_WAITS**: This contains information regarding locks that transactions are currently waiting on. This can be used to show lock relationships between different transactions. Full column details are available here: `http://dev.mysql.com/doc/innodb-plugin/1.0/en/innodb-information-schema-innodb_lock_waits.html`.

Global status and global variables

MySQL's global status and global variables output contains many useful troubleshooting data points. This output can be used to trend usage as well as report on database status. The primary goal of using `SHOW GLOBAL VARIABLES;` when troubleshooting is to find the value of a specific system variable, or ensure that different system variables are set to proper values as expected. To see InnoDB specific variables we can run the following query: `SHOW GLOBAL VARIABLES LIKE 'innodb%';`.

Using `SHOW GLOBAL STATUS;` will output information about runtime statistics. We can see InnoDB specific statistics by running the following query: `SHOW GLOBAL STATUS LIKE 'innodb%';`.

Third-party resources for trending and visualization

Several tools exist for watching trends of InnoDB usage, as well as for generating graphs to visualize the trends. The details of using each application are out of the scope of this book, however the application websites contain supporting documentation for use with MySQL servers. The following are some of the most commonly used tools for trending and visualization:

- **Cacti**: `http://www.cacti.net`
- **Graphite**: `https://launchpad.net/graphite`
- **MySQL Enterprise Monitor**: `http://www.mysql.com/products/enterprise/monitor.html`
- **MonYog**: `http://www.webyog.com/en/`
- **Kontrollbase**: `http://kontrollsoft.com/software-kontrollbase`
- **OpenTSDB**: `http://opentsdb.net`
- **Zabbix**: `http://www.zabbix.com`
- **Percona Toolkit**: `http://percona.com`
- **InnoTop**: `http://code.google.com/p/innotop/`

Equations for analyzing InnoDB statistics

The following code is generic pseudo code that you can apply to any programming language. The variable names correspond to the output values seen in `SHOW GLOBAL STATUS;` and `SHOW GLOBAL VARIABLES;`.

We can use the following equations and queries to generate analytics and reports about InnoDB usage and memory buffer utilization:

- `Innodb_buffer_pool_pages_ratio = Innodb_buffer_pool_pages_free/ Innodb_buffer_pool_pages_total`

  ```
  innodb_data_size = SELECT SUM(DATA_LENGTH) AS Value FROM
  INFORMATION_SCHEMA.TABLES WHERE ENGINE='innodb';
  innodb_index_size = SELECT SUM(INDEX_LENGTH) AS Value FROM
  INFORMATION_SCHEMA.TABLES WHERE ENGINE='innodb';
  ```

- Minimum size of `innodb_buffer_pool_size` to keep InnoDB indexes and data in the buffer pool equals to `innodb_index_size + innodb_data_size`

We can use these equations to determine MySQL memory usage. Again, the variables correspond to the values from the output of "SHOW GLOBAL STATUS;" and "SHOW GLOBAL VARIABLES;"

- `per_thread_buffers = (read_buffer_size + read_rnd_buffer_size + sort_buffer_size + thread_stack + join_buffer_size + binlog_cache_size) * max_connections`
- `per_thread_max_buffers = (read_buffer_size + read_rnd_buffer_size + sort_buffer_size + thread_stack + join_buffer_size + binlog_cache_size) * Max_used_connections`
- `global_buffers = innodb_buffer_pool_size + innodb_additional_mem_pool_size + innodb_log_buffer_size + key_buffer_size + query_cache_size`
- `maximum_memory_allocated = global_buffers + per_thread_max_buffers`
- `total_memory_allocation_allowed = global_buffers + per_thread_buffers`
- percent of system RAM used by MySQL = `(total_memory_allocation_allowed* 100) / os_mem_total`

To generate statistics about InnoDB engine data and index sizes, as well as comparison to MyISAM tables, for example, we can use the following SQL query to generate a report:

```
SELECT (data_size + index_size) / gb AS total_size_gb ,
index_size / gb AS index_size_gb , data_size / gb AS data_size_gb ,
SUM(innodb_index_size + innodb_data_size) / POW(1024,3) AS innodb_
total_size_gb ,
 innodb_data_size / POW(1024,3) AS innodb_data_size_gb , innodb_index_
size / POW(1024,3) AS innodb_index_size_gb,
SUM(myisam_index_size + myisam_data_size) / POW(1024,3) AS myisam_
total_size_gb,
 myisam_data_size / POW(1024,3) AS myisam_data_size_gb,
myisam_index_size / POW(1024,3) AS myisam_index_size_gb,
index_size / (data_size + index_size) * 100 AS perc_index,
data_size / (data_size + index_size) * 100 AS perc_data,
innodb_index_size / (innodb_data_size + innodb_index_size) * 100 AS
innodb_perc_index,
innodb_data_size / (innodb_data_size + innodb_index_size) * 100 AS
innodb_perc_data,
myisam_index_size / (myisam_data_size + myisam_index_size) * 100 AS
myisam_perc_index,
myisam_data_size / (myisam_data_size + myisam_index_size) * 100 AS
myisam_perc_data,
innodb_index_size / index_size * 100 AS innodb_perc_total_index,
innodb_data_size / data_size * 100 AS innodb_perc_total_data,
myisam_index_size / index_size * 100 AS myisam_perc_total_index,
myisam_data_size / data_size * 100 AS myisam_perc_total_data
FROM ( select SUM(data_length) data_size , SUM(index_length) index_
size , SUM(IF(engine = 'innodb', data_length, 0)) AS innodb_data_size
,
SUM(IF(engine = 'innodb', index_length, 0)) AS innodb_index_size,
SUM(IF(engine = 'myisam', data_length, 0)) AS myisam_data_size,
SUM(IF(engine = 'myisam', index_length, 0)) AS myisam_index_size,
POW(1024, 3) gb FROM INFORMATION_SCHEMA.TABLES ) a;
```

Summary

In this chapter we've covered typical maintenance procedures as well as common methods for querying runtime information and methods for reporting on InnoDB metrics. We've covered useful equations for generating statistical data as well as common applications that can be used for visualization and information trending.

6
Troubleshooting InnoDB

There are several situations that require manual inspection and troubleshooting of InnoDB:

- Performance or bottleneck problems
- Dead locks or wait-timeouts
- Process failure or system crashes

InnoDB is shipped with several useful methods to aid in the troubleshooting process via MySQL and status log files, information_schema querying, and direct reporting from the InnoDB engine.

Troubleshooting system issues

System crashes are never a positive event for a production server but the troubleshooting process doesn't have to be painful if one is aware of the processes ahead of time. InnoDB system failure issues can typically be attributed to hardware limitation or configuration errors where usable limits of buffers or logs were exceeded.

InnoDB system crashes

InnoDB, while typically a very stable database engine for high traffic transactional environments, is not without flaws. Sometimes buffers crash, unknown bugs occur, or other issues arise that can cause a database to suffer downtime. Luckily we can rely on ACID compliance to ensure that no data is lost in the event of a crash. If you have set InnoDB to use ACID compliant settings you will not lose data unless an external issue has occurred. External issues includes InnoDB thinking it has written data to disk but a Software RAID mechanism or RAID system using a non-battery backed cache, has not fully written changes to disk.

When InnoDB crashes from process failure the `ib_data` log files are not flushed to disk and some transactions may not have completed the `COMMIT` phase. Ideally, when MySQL is started after a crash, InnoDB will begin the crash recovery process and given enough time it will complete without errors.

We can see what state the crash recovery process is in by watching the MySQL error log.

```
$> sudo tail -f <data_dir>/<hostname>.err
```

While watching output we can see the steps InnoDB goes through for crash recovery:

1. Prior to accepting connections, InnoDB will start the log redo operation to apply any data changes that were not flushed from the buffer pool to the InnoDB tablespace files. If there are no changes waiting to be written then this step will be skipped. This will be referenced in the log files via timestamp and percentage-complete progress updates. If our server is configured with large redo logs or has relatively slow I/O resources then this process can take a significant amount of time. It is not uncommon for a highly active server with large logs to require thirty minutes or more to complete this process.
2. Transaction roll-back: Any transactions that did not complete prior to system failure will be issued a rollback so that data remains consistent.
3. Index page read and buffer merges: InnoDB will read the index pages and merge pending shared tablespace, insert buffer changes into the secondary indexes. This operation is executed in parallel.
4. Delete purge process: All data references that were marked for deletion but did not get removed prior to the crash will be purged to keep data consistent.

If InnoDB runs into any errors during this process we can reference the error code with the documentation and follow the recommendations for solving the issue. The output from this file can also be used for submitting a bug report if we run into a non documented issue.

Using InnoDB crash recovery modes

If InnoDB cannot complete the redo-log crash recovery process then the data recovery process must be initiated. This involves forcing InnoDB to start in one of several optional states in order to retrieve our data. These recovery modes do not fix the underlying issue(s) that caused the crash nor do they fix InnoDB to the state that it can be restarted and continue to operate per usual. The recovery modes simply allow us to log in to MySQL and export data from our InnoDB tables so that we can import that data into a correctly operating InnoDB instance elsewhere, or after MySQL has been fixed so that it can start correctly.

InnoDB offers several levels of operational limitation to aid in the data recovery process. If InnoDB is started via any of these modes it will ignore any attempts at changing or adding data via `INSERT`, `UPDATE`, or `DELETE` queries. Our only choices for interacting with the database are via `SELECT`, `DROP`, and `CREATE`. Ideally, during the data recovery process, it is preferable to start at the lowest recovery mode and attempt to recover the data; if data cannot be recovered or InnoDB fails to start at the lower setting then we increase the value and try again. In some cases we can determine, through a process of elimination, what is wrong with InnoDB by attempting to start the engine with different recovery modes while watching the error log and comparing the progress to other recovery modes. The following are some of the InnoDB crash recovery modes:

- `1 SRV_FORCE_IGNORE_CORRUPT`: This allows the server to start even if there are corrupt pages
- `2 SRV_FORCE_NO_BACKROUND`: This starts the server without the main InnoDB thread running
- `3 SRV_FORCE_NO_TRX_UNDO`: This skips transaction roll-back process during initialization
- `4 SRV_FORCE_NO_IBUF_MERGE`: This skips the insert buffer merge operations during initialization and does not generate statistics for any InnoDB tables
- `5 SRV_FORCE_NO_UNDO_LOG_SCAN`: This ignores the redo-log read + apply process during initialization and treats all transactions as committed even if they had not yet completed
- `6 SRV_FORCE_NO_LOG_REDO`: This skips the redo-log process during initialization

Enabling InnoDB recovery modes

The process for enabling recovery mode is as follows:

1. Ensure that MySQL is offline and no child processes are running.
2. Edit the main `my.cnf` file and add or change the value of `innodb_force_recovery = N` where `N` is one of the previously mentioned states.
3. Watch the MySQL error log in one terminal via the `tail -f` command.
4. Start the MySQL Server process via the command line while running as the `mysql` user. Do not use the operating system service controls to start MySQL: `sudo -u mysql mysqld`, this will allow us to see all output of the process while it goes through the recovery steps.

5. Once the MySQL server process has started up and enabled us to log in we can then go about exporting our data either via `SELECT INTO OUTFILE` statements or other forms of backing up data to the filesystem, such as `mysqldump`.
6. Optionally, if we do not have an additional server to migrate the data to, and we have time during the downtime window to re-import the data, then we can attempt to fix the problem table(s) after we have a valid SQL dump file of its contents. In this method we follow these processes:
    1. Complete the first five steps above to generate a SQL dump file per each table needed.
    2. Issue a `DROP` command for the corrupt table(s).
    3. Shut down MySQL, then remove the force recovery option or comment it out.
    4. Start the MySQL Server process. If the server starts correctly without errors and we can log in then it is safe to re-create the table(s) and import their data.

Utilizing InnoDB status statistics

InnoDB has many advanced statistics available that we can use for troubleshooting and performance monitoring. The simplest method of querying this information is to execute the query following and read the output:

```
mysql> pager less;
mysql> SHOW ENGINE INNODB STATUS\G
```

This information is also periodically written to the InnoDB status file if you have enabled the configuration variable for InnoDB status file: `innodb_status_file`.

In order to record these statistics in a periodic manner, we need to create several tables. These tables do not store any data, but when we execute the commands InnoDB recognizes them as a signal to begin writing statistics to the MySQL error log. These tables can be created in any database, but best practice is to create them in their own schema so that they do not conflict with application specific schemas. InnoDB requires one table per monitor, so we will execute the following queries to create the tables:

```
mysql> CREATE DATABASE innodb_monitors; USE innodb_monitors;
mysql> CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
mysql> CREATE TABLE innodb_lock_monitor (a INT) ENGINE=INNODB;
mysql> CREATE TABLE innodb_tablespace_monitor (a INT) ENGINE=INNODB;
mysql> CREATE TABLE innodb_table_monitor (a INT) ENGINE=INNODB;
```

If we need to stop the monitors from writing to the error log we can issue a corresponding `DROP TABLE` query for each monitor that needs to be disabled. If the tables are not dropped then the monitors will continue to run until MySQL has been stopped. Upon MySQL restart, the InnoDB monitors do not start again even if the tables exist. The `CREATE TABLE` statements must be executed again if we wish to start the monitors. The following commands are used to stop the monitors and remove the tables:

- `mysql> USE innodb_monitors;`
- `mysql> DROP TABLE innodb_lock_monitor;`
- `mysql> DROP TABLE innodb_monitor;`
- `mysql> DROP TABLE innodb_monitors;`
- `mysql> DROP TABLE innodb_table_monitor;`
- `mysql> DROP TABLE innodb_tablespace_monitor;`

Monitoring tables should only be enabled when specifically needed for troubleshooting purposes. On a very active server the periodic information, in particular the lock monitor output, will quickly fill up the error log which will uneccesarily use I/O resources if the logged information is not being used. Additionally, InnoDB does take a small performance hit when the monitors are enabled, as it is processing and writing information that consumes CPU cycles and memory that would otherwise be allocated to processing transactions.

For more information about InnoDB statistics and monitoring you can refer to the MySQL manual section titled: *14.3.14.2. SHOW ENGINE INNODB STATUS* and the InnoDB Monitors: `http://dev.mysql.com/doc/refman/5.5/en/innodb-monitors.html`.

Troubleshooting InnoDB performance

In many cases we can point to hardware resources as the bottlenecks of database performance issues, and the solution in those cases is to upgrade hardware to increase available CPU cycles, add more RAM for memory buffers, or faster disk systems for increased IOPS workloads. Query optimization is another common task for improving performance and should be addressed in conjunction with hardware and InnoDB tuning.

However, performance bottlenecks can arise from different issues when using InnoDB and are not always hardware or query related. Incorrect configuration values for your application's workload or failing to change the default InnoDB settings on a server are common causes of poor InnoDB performance.

The following are baseline variables that can be changed from their default values to help prevent performance bottlenecks. These variables should also be inspected if you encounter performance issues, as they are primary methods for altering resource utilization by InnoDB. The configuration variable tuning process is covered in *Chapter 2, Basic Configuration Parameters* and *Chapter 3, Advanced Configuration Parameters*.

I/O resources

- `innodb_io_capacity`
- `innodb_file_io_threads` (5.1 only)
- `innodb_write_io_threads` (5.5 and greater)
- `innodb_read_io_threads` (5.5 and greater)
- `innodb_flush_method`
- `innodb_flush_log_at_trx_commit`
- `skip-innodb-doublewrite`
- `sync_binlog`

Table-space and logging

- `innodb_data_file_path`
- `innodb_log_file_size`
- `innodb_log_files_in_group`
- `innodb_file_per_table`

Memory resources

- `innodb_buffer_pool_size`
- `innodb_buffer_pool_instances`
- `innodb_additional_mem_pool_size`
- `innodb_log_buffer_size`

Concurrency settings

- `innodb_thread_concurrency`
- `innodb_commit_concurrency`

Global status variables – trending statistics

The following variables can be analyzed and monitored over time for trending purposes. When viewing the output of these metrics via time-series graphs, we can see relationships and correlations, which are useful for aligning service and performance issues with the health and status of the InnoDB engine.

Collecting the output of these variables is done via one of the two types of queries:

```
mysql> show global status;
mysql> show global status like 'innodb_%';
mysql> show global status like '$variable';
```

InnoDB performance status variables

The following variables can be inspected and monitored to gain insight into the ongoing performance of InnoDB. The following documentation offers details for each variable: `http://dev.mysql.com/doc/refman/5.5/en/server-status-variables.html`.

- `Innodb_buffer_pool_pages_data`
- `Innodb_buffer_pool_pages_dirty`
- `Innodb_buffer_pool_pages_total`
- `Innodb_buffer_pool_read_requests`
- `Innodb_buffer_pool_reads`
- `Innodb_buffer_pool_write_requests`
- `Innodb_data_fsyncs`
- `Innodb_data_reads`
- `Innodb_data_writes`
- `Innodb_pages_read`
- `Innodb_pages_written`
- `Innodb_row_lock_time_avg`
- `Innodb_rows_deleted`
- `Innodb_rows_inserted`
- `Innodb_rows_read`
- `Innodb_rows_updated`

InnoDB pending operation status variables

The following can be monitored and trended to analyze operations that are waiting on other resources, that is, in a pending state. The following documentation offers details for each of the mentioned variables: `http://dev.mysql.com/doc/refman/5.5/en/server-status-variables.html`.

- `Innodb_log_waits`
- `Innodb_os_log_pending_fsyncs`
- `Innodb_os_log_pending_writes`
- `Innodb_data_pending_fsyncs`
- `Innodb_data_pending_reads`
- `Innodb_data_pending_writes`
- `Innodb_row_lock_current_waits`
- `Innodb_row_lock_waits`

Get to grips with InnoDB error codes

During an operation of InnoDB, you may encounter specific error codes due to a conflict or illegal operation. The more common ones are explained here. Due to the nature of Foreign Key design, many of the most common errors result in lack of application of the rules required for Foreign Key operation.

1005 (ER_CANT_CREATE_TABLE)

This is a table creation failed error. Possible errors include a reference to (-1); a column name in the table being created conflicts with change; a to an a existing table in the InnoDB data dictionary, and (150) FK constraint was not properly defined.

1016 (ER_CANT_OPEN_FILE)

InnoDB data file is missing and cannot be opened for operation. Typically a `.frm` file exists for this table but the main data file is not available. This can generally be solved by deleting the `.frm` file manually and re-creating the table from a backup. See the manual for full details.

1114 (ER_RECORD_FILE_FULL)

This error indicates that the tablespace file in use has run out of space. The solution is to add an additional tablespace file or add the `autoextend` option to the existing tablespace. If you have configured InnoDB to use RAW partitions for tablespaces then you will need to configure an additional tablespace.

1205 (ER_LOCK_WAIT_TIMEOUT)

If you find this error during a transaction execution it means the lock wait time has been exceeded and the transaction has been rolled back. If you are aware of query execution statistics for your application, it is possible to increase the `innodb_lock_wait_timeout` value to prevent transactions from exceeding the timeout. Keep in mind that increasing the timeout value is not necessarily a good idea for all applications, as the original issue at hand is commonly a problematic long-running transaction that is locking data that other queries need access to. Addressing these issues by increasing the timeout will not solve the underlying execution issue of problematic long-running queries.

1206 (ER_LOCK_TABLE_FULL)

Commonly seen in cases where the database server has been configured with a small `innodb_buffer_pool_size` value, this error is a result of the lock table size not being large enough to support the concurrency of your database traffic in regard to transaction lock handling. The solution depends on the underlying queries; if you are getting this error on a server with a large buffer pool but only when issuing very large `INSERT` transactions that requires a large number of locks, and you cannot increase the size of the buffer pool then you may be able to issue several smaller transactions to accomplish the same changes, thus requiring fewer locks per transaction. If you cannot split the query into smaller transactions then you must increase the size of the buffer pool.

1213 (ER_LOCK_DEADLOCK)

The transaction in question has encountered a deadlock and must be run again to complete. A deadlock, essentially a catch-22 for transactions, occurs when transaction A is waiting on resources or locks that transaction B is holding, except that transaction B is waiting on a lock that A is holding; thus neither query will execute because they are waiting for each other to finish and are in a perpetual loop. Eventually the transactions will timeout and have to be executed again to complete. To help prevent deadlocks be sure you have InnoDB set up for ACID compliance.

1216 (ER_NO_REFERENCED_ROW)

This error occurs when a query is executed that holds a foreign key that references a parent row, but the parent row does not exist. Generally this error will be seen if you are restoring backup data out of sync and have not disabled key checking during the import process. The solution for an active database is to have the parent row inserted beforehand, and retry the transaction. The solution for a non active database or one that is being restored, is to either restore data in sequence or to disable key checking during the operation.

1217 (ER_ROW_IS_REFERENCED)

This error occurs when a row is requested for deletion but the row has referenced foreign key child rows, and you have not configured a cascading delete in the table definition. The solution is to either delete the child rows before the parent row or to issue an alter table command to allow for a cascading delete that will hand foreign-key relationships, then re-issue your delete transactions.

This table foreign-key statement will result in `ERROR 1217` when encountering parent/child delete relationships: `FOREIGN KEY (id) REFERENCES referenced_table(child_id);`.

This table foreign-key statement will handle cascading deletes: `FOREIGN KEY (id) REFERENCES referenced_table(child_id) ON DELETE CASCADE;`.

InnoDB troubleshooting and backup applications

This is not a comprehensive list, but contains commonly used tools that help troubleshooting, backup, and repair of InnoDB tables.

- **InnoCheckSum** (`http://dev.mysql.com/doc/refman/5.5/en/innochecksum.html`): It allows the user to generate and analyze data checksum values for their InnoDB data. This is useful for identifying damaged or faulty data.
- **InnoDB Tools** (`http://code.google.com/p/innodb-tools/`): It is a collection of tools that helps the user recover data from damaged, corrupt, or otherwise broken InnoDB tables.

- **InnoDB Hot Backup** (`http://www.innodb.com/doc/hot_backup/manual.html`): It allows a user to generate a backup file for their database while the server is still running, thus allowing traffic to continue while the backup is running. It has largely been replaced by MySQL Enterprise Backup. This is particularly useful if one has started the database in recovery mode and needs to get a live snapshot of data for seeding a replacement server.
- **ZRM Recovery Manager** (`http://www.zmanda.com/backup-mysql.html`): This is a full web-based backup management system that leverages a hot-backup capable client and extensive enterprise features.
- **R1Soft Enterprise Backup** (`http://r1soft.idera.com/server-backup-enterprise`): It offers enterprise-wide database backup solutions with support for bare-metal database restoration. InnoDB backups are handled by a kernel module agent that allows completely transparent backups while MySQL is actively running.
- **MySQL Enterprise Backup** (`http://www.mysql.com/products/enterprise/backup.html`): It offers similar features to InnoDB Hot Backup but with ongoing support from the Oracle corporation as well as offering additional features.
- **MyDumper** (`http://www.mydumper.org`): A tool from several former members of the MySQL Support team, MyDumper is a multi-threaded backup and restore tool that supports InnoDB, MyISAM, and Drizzle database tables.
- **Percona XtraBackup** (`http://www.percona.com/doc/percona-xtrabackup/`): A simple command line based tool for online backups of InnoDB tables. Offers streaming and incremental backup support.

Summary

In this chapter we have covered the basic methods for troubleshooting common InnoDB issues. This is a particularly active topic in MySQL forums and on the Internet in general, due to the popularity of InnoDB and the complexity of its configurations. Since there isn't a simple one size fits all setup for InnoDB and additionally due to the ultra conservative default settings that InnoDB ships with, it is not uncommon for performance issues to arise in production environments, but remember that the best solution is preventative maintenance, sound configuration, and consistent monitoring. In the next chapter, the conclusion of the book, we'll cover reference material and discuss useful resource links for InnoDB information on the Internet.

7
References and Links

MySQL's stability and enterprise features, many provided solely by InnoDB, continues to make it the go-to relational database of the last decade. Its growth has only increased over the years, as InnoDB developers have added additional features while maintaining a consistent release schedule that addresses bug reports and functionality requests from the user community. If trends continue as they have, we may see InnoDB become the most actively used open source transactional database engine of the modern Internet era.

General documentation

The following sites contain a well-rounded amount of reference documentation necessary for the daily life of a MySQL DBA or developer. Additionally, there are many blogs available online that focus primarily on MySQL database development and administration, which can be viewed on the aggregation site "Planet MySQL".

- `http://dev.mysql.com/doc/`
- `http://www.innodb.com/wp/support/documentation/`
- `http://www.innodb.com/doc/hot_backup/manual.html`
- `http://en.wikipedia.org/wiki/Comparison_of_MySQL_database_engines`
- `http://bugs.mysql.com/`
- `http://planet.mysql.com/`

Reference books

MySQL is a growing technology and as such it has spawned a fair number of titles referencing the subject. There are books covering MySQL specializations for system engineering, developers, web programmers, as well as database administrators. The following are some of the more common choices:

- *MySQL Cookbook, Paul DuBois*
- *MYSQL in a Nutshell (In a Nutshell), Russell J. T. Dyer, O'Reilly Publications*
- *MySQL High Availability: Tools for Building Robust Data Centers, Charles Bell, Mats Kindahl and Lars Thalmann*
- *High Performance MySQL: Optimization, Backups, and Replication, Baron Schwartz, Peter Zaitsev and Vadim Tkachenko, (Mar 30, 2012)*
- *MySQL (5th Edition), Paul DuBois*
- *Effective MySQL Replication Techniques in Depth, Ronald Bradford and Chris Schneider*
- *High Availability MySQL Cookbook, Alex Davies*
- *MySQL for Python, Albert Lukaszewski*
- *MySQL Admin Cookbook, Daniel Schneller and Udo Schwedt*
- *Creating your MySQL Database: Practical Design Tips and Techniques, Marc Delisle*

Terminology

The technical world is quite full of acronyms and there was no shortage of them in the former pages. The following links will show you the most commonly used acronyms used in the book:

- `http://en.wikipedia.org/wiki/MySQL`
- `http://en.wikipedia.org/wiki/InnoDB`
- `http://en.wikipedia.org/wiki/Multiversion_concurrency_control`
- `http://en.wikipedia.org/wiki/ACID`
- `http://en.wikipedia.org/wiki/Database_transaction`
- `http://en.wikipedia.org/wiki/Isolation_(database_systems)`
- `http://en.wikipedia.org/wiki/Lock_(database)`
- `http://en.wikipedia.org/wiki/Database_normalization`
- `http://en.wikipedia.org/wiki/Denormalization`

- http://en.wikipedia.org/wiki/NoSQL
- http://en.wikipedia.org/wiki/Relational_database_management_system
- http://en.wikipedia.org/wiki/IOPS

InnoDB tuning references

- http://dev.mysql.com/doc/refman/5.5/en/optimizing-innodb.html
- http://developers.sun.com/solaris/articles/mysql_perf_tune.html
- https://blogs.oracle.com/luojiach/entry/mysql_innodb_performance_tuning_for
- http://www.innodb.com/wp/wp-content/uploads/2007/05/uc2007-innodb-performance-optimization.pdf
- http://highscalability.com/blog/category/mysql
- http://www.mysqlperformanceblog.com/

Index

S

T

Z

Thank you for buying
Instant InnoDB

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

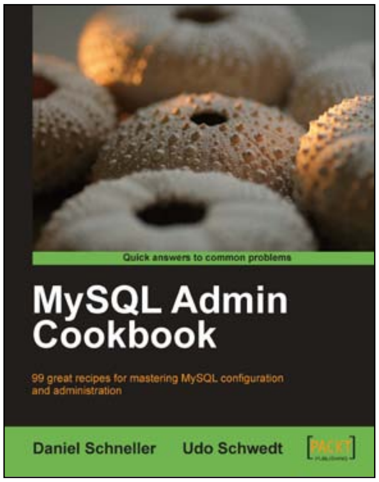
Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] PUBLISHING



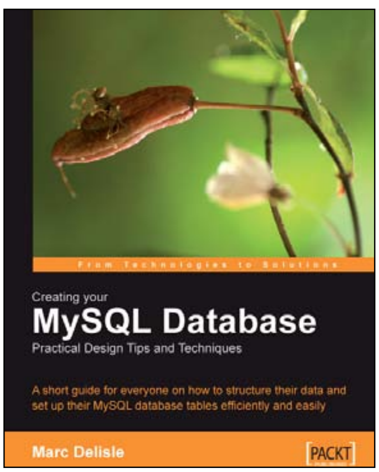
MySQL Admin Cookbook

ISBN: 978-1-84719-796-2 Paperback: 376 pages

99 great recipes for mastering MySQL configuration and administration

1. Set up MySQL to perform administrative tasks such as efficiently managing data and database schema, improving the performance of MySQL servers, and managing user credentials
2. Deal with typical performance bottlenecks and lock-contention problems
3. Restrict access sensibly and regain access to your database in case of loss of administrative user credentials

---



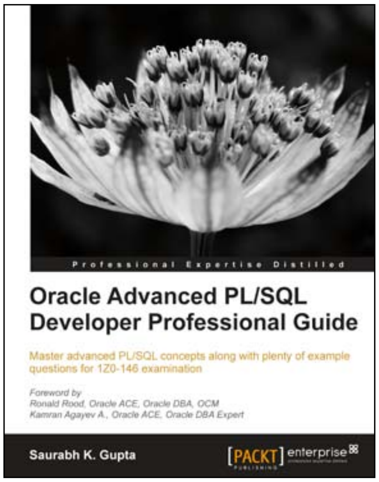
Creating your MySQL Database: Practical Design Tips and Techniques

ISBN: 978-1-90481-130-5 Paperback: 108 pages

A short guide for everyone on how to structure your data and set-up your MySQL database tables efficiently and easily

1. How best to collect, name, group, and structure your data
2. Design your data with future growth in mind
3. Practical examples from initial ideas to final designs
4. The quickest way to learn how to design good data structures for MySQL

Please check **www.PacktPub.com** for information on our titles



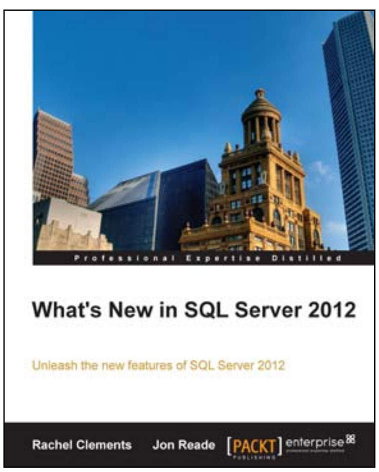
Oracle Advanced PL/SQL Developer Professional Guide

ISBN: 978-1-84968-722-5 Paperback: 440 pages

Master advanced PL/SQL concepts along with plenty of example questions for 1Z0-146 examination

1. Blitz the 1Z0-146 exam
2. Master the advanced features of PL/SQL to design and optimize code using real-time demonstrations
3. Efficiently design PL/SQL code with cursor design and subtypes

---



What's New in SQL Server 2012

ISBN: 978-1-84968-734-8 Paperback: 238 pages

Unleash the new features of SQL Server 2012

1. Upgrade your skills to the latest version of SQL Server
2. Discover the new dimensional model in Analysis Services
3. Utilize data alerts and render reports to the latest versions of Excel and Word
4. Build packages to leverage the new features in the Integration Services environment
5. Plan, load and simulate real-world database activity using Distributed Replay

Please check **www.PacktPub.com** for information on our titles