华章科技

畅销书全新升级，第1版广获好评；资深MySQL专家撰写，全球知名MySQL数据库服务提供商Percona公司CTO作序推荐，国内多位数据库专家联袂推荐

基于MySQL 5.6，结合源代码，从存储引擎内核角度对InnoDB的整体架构、核心实现和工作机制进行深入剖析

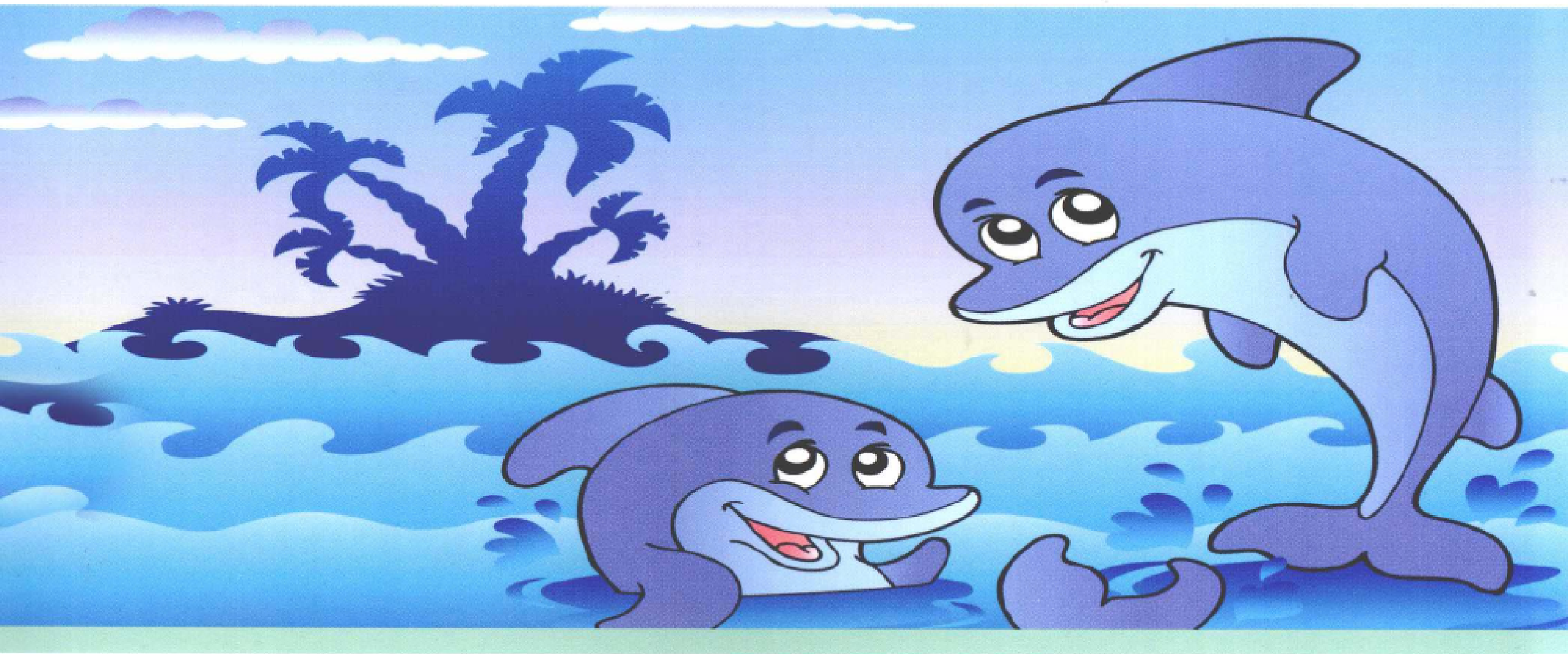
数据库技术丛书

Inside MySQL: InnoDB Storage Engine, Second Edition

MySQL技术内幕

InnoDB存储引擎

第2版



姜承尧◎著

机械工业出版社
China Machine Press

数据库技术丛书

Inside MySQL: InnoDB Storage Engine, Second Edition

MySQL技术内幕

InnoDB存储引擎

第2版

姜承尧◎著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

MySQL 技术内幕：InnoDB 存储引擎 / 姜承尧著 .—2 版 . —北京：机械工业出版社，2013.6
（数据库技术丛书）

ISBN 978-7-111-42206-8

I. M… II. 姜… III. 关系数据库系统 IV. TP311.138

中国版本图书馆 CIP 数据核字（2013）第 079001 号

本书由国内资深 MySQL 专家亲自执笔，国内外多位数据库专家联袂推荐。作为国内唯一一本关于 InnoDB 的专著，本书的第 1 版广受好评，第 2 版不仅针对最新的 MySQL 5.6 对相关内容进行了全面的补充，还根据广大读者的反馈意见对第 1 版中存在的不足进行了完善，全书大约重写了 50%的内容。本书从源代码的角度深度解析了 InnoDB 的体系结构、实现原理、工作机制，并给出了大量最佳实践，能帮助你系统而深入地掌握 InnoDB，更重要的是，它能为你设计管理高性能、高可用的数据库系统提供绝佳的指导。

全书一共 10 章，首先宏观地介绍了 MySQL 的体系结构和各种常见的存储引擎以及它们之间的比较；接着以 InnoDB 的内部实现为切入点，逐一详细讲解了 InnoDB 存储引擎内部的各个功能模块的实现原理，包括 InnoDB 存储引擎的体系结构、内存中的数据结构、基于 InnoDB 存储引擎的表和页的物理存储、索引与算法、文件、锁、事务、备份与恢复，以及 InnoDB 的性能调优等重要的知识；最后对 InnoDB 存储引擎源代码的编译和调试做了介绍，对大家阅读和理解 InnoDB 的源代码有重要的指导意义。

本书适合所有希望构建和管理高性能、高可用性的 MySQL 数据库系统的开发者和 DBA 阅读。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：姜 影
北京市荣盛彩色印刷有限公司印刷
2013 年 5 月第 2 版第 1 次印刷
186mm×240mm · 27.25 印张
标准书号：ISBN 978-7-111-42206-8
定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

推 荐 序

It's fair to say that MySQL is the most popular open source database. It has a very large installed base and number of users. Let's see what are the reasons MySQL is so popular, where it stands currently, and maybe touch on some of its future (although predicting the future is rarely successful).

Looking at the customer area of MySQL, which includes Facebook, Flickr, Adobe (in Creative Suite 3), Drupal, Digg, LinkedIn, Wikipedia, eBay, YouTube, Google AdSense (source http://mysql.com/customers/ and public resources), it's obvious that MySQL is everywhere. When you log in to your popular forum (powered by Bulleting) or blog (powered by WordPress), most likely it has MySQL as its backend database. Traditionally, two MySQL's characteristics, simplicity of use and performance, were what allowed it to gain such popularity. In addition to that, availability on a very wide range of platforms (including Windows) and built-in replication, which provides an easy scale-out solution for read-only clients, gave more user attractions and production deployments. There is simple evidence of MySQL's simplicity: In 15 minutes or less, you really can get installed, have a working database, and start running queries and store data. From its early stages MySQL had a good interface to most popular languages for Web development - PHP and Perl, and also Java and ODBC connectors.

There are two best known storage engines in MySQL: MyISAM and InnoDB (I don't cover NDB cluster here; it's a totally different story). MyISAM comes as the default storage engine and historically it is the oldest, but InnoDB is ACID compliant and provides transactions, row-level locking, MVCC, automatic recovery and data corruption detection. This makes it the storage engine you want to choose for your application. Also, there is the third-party transaction storage engine PBXT, with characteristics similar to InnoDB, which is included in the MariaDB distribution.

MySQL's simplicity has its own drawback. Just as it is very easy to start working with it, it is very easy to start getting into trouble with it. As soon as your website or forum gets popular, you may figure out that the database is a bottleneck, and that you need special skills and tools to fix it.

The author of this book is a MySQL expert, especially in InnoDB storage engineB. Hence, I highly recommend this book to new users of InnoDB as well as uers who already have well-tuned InnoDB-based applications but need to get internal out of them.

Vadim Tkachenko
全球知名 MySQL 数据库服务提供商 Percona 公司 CTO
知名 MySQL 数据库博客 MySQLPerformanceBlog.com 作者
《高性能 MySQL（第 2 版）》作者之一

前　言

为什么要写这本书

过去这些年我一直在和各种不同的数据库打交道，见证了 MySQL 从一个小型的关系型数据库发展为各大企业的核心数据库系统的过程，并且参与了一些大大小小的项目的开发工作，成功地帮助开发人员构建了可靠的、健壮的应用程序。在这个过程中积累了一些经验，正是这些不断累积的经验赋予了我灵感，于是有了这本书。这本书实际上反映了这些年来我做了哪些事情，其中汇集了很多同行每天可能都会遇到的一些问题，并给出了解决方案。

MySQL 数据库独有的插件式存储引擎架构使其和其他任何数据库都不同。不同的存储引擎有着完全不同的功能，而 InnoDB 存储引擎的存在使得 MySQL 跃入了企业级数据库领域。本书完整地讲解了 InnoDB 存储引擎中最重要的一些内容，即 InnoDB 的体系结构和工作原理，并结合 InnoDB 的源代码讲解了它的内部实现机制。

本书不仅讲述了 InnoDB 存储引擎的诸多功能和特性，还阐述了如何正确地使用这些功能和特性，更重要的是，还尝试了教我们如何 Think Different。Think Different 是 20 世纪 90 年代苹果公司在其旷日持久的宣传活动中提出的一个口号，借此来重振公司的品牌，更重要的是，这个口号改变了人们对技术在日常生活中的作用的看法。需要注意的是，苹果的口号不是 Think Differently，是 Think Different，Different 在这里做名词，意味该思考些什么。

很多 DBA 和开发人员都相信某些“神话”，然而这些“神话”往往都是错误的。无论计算机技术发展的速度变得多快，数据库的使用变得多么简单，任何时候 Why 都比 What 重要。只有真正理解了内部实现原理、体系结构，才能更好地去使用。这正是人类正确思考问题的原则。因此，对于当前出现的技术，尽管学习其应用很重要，但更重要的是，应当正确地理解和使用这些技术。

关于本书，我的头脑里有很多个目标，但最重要的是想告诉大家如下几个简单的观点：

- 不要相信任何的“神话”，学会自己思考；
- 不要墨守成规，大部分人都知道的事情可能是错误的；
- 不要相信网上的传言，去测试，根据自己的实践做出决定；
- 花时间充分地思考，敢于提出质疑。

当前有关 MySQL 的书籍大部分都集中在教读者如何使用 MySQL，例如 SQL 语句的使用、复制的搭建的、数据的切分等。没错，这对快速掌握和使用 MySQL 数据库非常有好处，但是真正的数据库工作者需要了解的不仅仅是应用，更多的是内部的具体实现。

MySQL 数据库独有的插件式存储引擎使得想要在一本书内完整地讲解各个存储引擎变得十分困难，有的书可能偏重对 MyISAM 的介绍，有的可能偏重对 InnoDB 存储引擎的介绍。对于初级的 DBA 来说，这可能会使他们的理解变得更困难。对于大多数 MySQL DBA 和开发人员来说，他们往往更希望了解作为 MySQL 企业级数据库应用的第一存储引擎的 InnoDB，我想在本书中，他们完全可以找到他们希望了解的内容。

再强调一遍，任何时候 Why 都比 What 重要，本书从源代码的角度对 InnoDB 的存储引擎的整个体系架构的各个组成部分进行了系统的分析和讲解，剖析了 InnoDB 存储引擎的核心实现和工作机制，相信这在其他书中是很难找到的。

第 1 版与第 2 版的区别

本书是第 2 版，在写作中吸收了读者对上一版内容的许多意见和建议，同时对于最新 MySQL 5.6 中许多关于 InnoDB 存储引擎的部分进行了详细的解析与介绍。希望通过这些改进，给读者一个从应用到设计再到实现的完整理解，弥补上一版中深度有余，内容层次不够丰富、分析手法单一等诸多不足。

较第 1 版而言，第 2 版的改动非常大，基本上重写了 50% 的内容。其主要体现在以下几个方面，希望读者能够在阅读中体会到。

- 本书增加了对最新 MySQL 5.6 中的 InnoDB 存储引擎特性的介绍。MySQL 5.6 版本是有史以来最大的一次更新，InnoDB 存储引擎更是添加了许多功能，如多线程清理线程、全文索引、在线索引添加、独立回滚段、非递归死锁检测、新的刷新算法、新的元数据表等。读者通过本书可以知道如何使用这些特性、新特性存在的局限性，并明白新功能与老版本 InnoDB 存储引擎之间实现的区别，从而在实际应用中充分利用这些特性。
- 根据读者的要求对于 InnoDB 存储引擎的 redo 日志和 undo 日志进行了详细的分析。读者应该能更好地理解 InnoDB 存储引擎事务的实现。在 undo 日志分析中，通过 InnoSQL 自带的元数据表，用户终于可对 undo 日志进行统计和分析，极大提高了 DBA 对于 InnoDB 存储引擎内部的认知。

- 对第 6 章进行大幅度的重写，读者可以更好地理解 InnoDB 存储引擎特有的 next-key locking 算法，并且通过分析锁的实现来了解死锁可能产生的情况，以及 InnoDB 存储引擎内部是如何来避免死锁问题的产生的。
- 根据读者的反馈，对 InnoDB 存储引擎的 insert buffer 模块实现进行了更为详细的介绍，读者可以了解其使用方法以及其内部的实现原理。此外还增加了对 insert buffer 的升级版本功能——change buffer 的介绍。

读者对象

本书不是一本面向应用的数据库类书籍，也不是一本参考手册，更不会教你如何在 MySQL 中使用 SQL 语句。**本书面向那些使用 MySQL InnoDB 存储引擎作为数据库后端开发应用程序的开发者和有一定经验的 MySQL DBA**。书中的大部分例子都是用 SQL 语句来展示关键特性的，如果想通过本书来了解如何启动 MySQL、如何配置 Replication 环境，可能并不能如愿。不过，在本书中，你将知道 InnoDB 存储引擎是如何工作的，它的关键特性的功能和作用是什么，以及如何正确配置和使用这些特性。

如果你想更好地使用 InnoDB 存储引擎，如果你想让你的数据库应用获得更好的性能，就请阅读本书。从某种程度上讲，技术经理或总监也要非常了解数据库，要知道数据库对于企业的重要性。如果技术经理或总监想安排员工参加 MySQL 数据库技术方面的培训，完全可以利用本书来“充电”，相信你一定不会失望的。

要想更好地学习本书的内容，要求具备以下条件：

- 掌握 SQL。
- 掌握基本的 MySQL 操作。
- 接触过一些高级语言，如 C、C++、Python 或 Java。
- 对一些基本算法有所了解，因为本书会分析 InnoDB 存储引擎的部分源代码，如果你能看懂这些算法，这会对你的理解非常有帮助。

如何阅读本书

本书一共有 10 章，每一章都像一本“迷你书”，可以单独成册，也就说你完全可以从书中任何一章开始阅读。例如，要了解第 10 章中的 InnoDB 源代码编译和调试的知识，就不必先去阅读第 3 章有关文件的知识。当然，如果你不太确定自己是否已经对本书所涉及的内容

完全掌握了，建议你系统性地阅读本书。

本书不是一本入门书籍，不会一步步引导你去如何操作。倘若你尚不了解 InnoDB 存储引擎，本书对你来说可能就显得沉重一些，建议你先查阅官方的 API 文档，大致掌握 InnoDB 的基础知识，然后再来学习本书，相信你会领略到不同的风景。

为了便于大家阅读，本书在提供源代码下载（下载地址：www.hzbook.com）的同时也将源代码附在了书中，因此占去了一些篇幅，还请大家理解。

勘误和支持

由于作者对 InnoDB 存储引擎的认知水平有限，再加上写作时可能存在疏漏，书中还存在许多需要改进的地方。在此，欢迎读者朋友们指出书中存在的问题，并提出指导性意见，不甚感谢。如果大家有任何与本书相关的内容需要与我探讨，请发邮件到 jiangchengyao@gmail.com，或者通过新浪微博 @insidemysql 与我联系，我会及时给予回复。最后，衷心地希望本书能给大家带来帮助，并祝大家阅读愉快！

致谢

在编写本书的过程中，我得到了很多朋友的热心帮助。首先要感谢 Pecona 公司的 CEO Peter Zaitsev 和 CTO Vadim Tkachenko，通过和他们的不断交流，使我对 InnoDB 存储引擎有了更进一步的了解，同时知道了怎样才能正确地将 InnoDB 存储引擎的补丁应用到生产环境。

其次，要感谢网易公司的各位同事们，能在才华横溢、充满创意的团队中工作我感到非常荣幸和兴奋。也因为在这个开放的工作环境中，我可以不断进行研究和创新。

此外，我还要感谢我的母亲，写本书不是一件容易的事，特别是这本书还想传达一些思想，在这个过程中我遇到了很多的困难，感谢她在这个过程中给予我的支持和鼓励。

最后，一份特别的感谢要送给本书的策划编辑杨福川和姜影，他们使得本书变得生动和更具有灵魂。此外还要感谢出版社的其他默默工作的同事们。

姜承尧

目　录

第 1 章　MySQL 体系结构和存储引擎

MySQL 被设计为一个可移植的数据库，几乎在当前所有系统上都能运行，如 Linux，Solaris、FreeBSD、Mac 和 Windows。尽管各平台在底层（如线程）实现方面都各有不同，但是 MySQL 基本上能保证在各平台上的物理体系结构的一致性。因此，用户应该能很好地理解 MySQL 数据库在所有这些平台上是如何运作的。

1.1　定义数据库和实例

在数据库领域中有两个词很容易混淆，这就是“数据库”（database）和“实例”（instance）。作为常见的数据库术语，这两个词的定义如下。

- **数据库**：物理操作系统文件或其他形式文件类型的集合。在 MySQL 数据库中，数据库文件可以是 frm、MYD、MYI、ibd 结尾的文件。当使用 NDB 引擎时，数据库的文件可能不是操作系统上的文件，而是存放于内存之中的文件，但是定义仍然不变。
- **实例**：MySQL 数据库由后台线程以及一个共享内存区组成。共享内存可以被运行的后台线程所共享。需要牢记的是，数据库实例才是真正用于操作数据库文件的。

这两个词有时可以互换使用，不过两者的概念完全不同。在 MySQL 数据库中，实例与数据库的关通常系是一一对应的，即一个实例对应一个数据库，一个数据库对应一个实例。但是，在集群情况下可能存在一个数据库被多个数据实例使用的情况。

MySQL 被设计为一个单进程多线程架构的数据库，这点与 SQL Server 比较类似，但与 Oracle 多进程的架构有所不同（Oracle 的 Windows 版本也是单进程多线程架构的）。这也就是说，**MySQL 数据库实例在系统上的表现就是一个进程。**

在 Linux 操作系统中通过以下命令启动 MySQL 数据库实例，并通过命令 ps 观察 MySQL 数据库启动后的进程情况：

```
[root@xen-server bin]# ./mysqld_safe&

[root@xen-server bin]# ps -ef | grep mysqld
```

```
root      3441  3258  0 10:23 pts/3    00:00:00 /bin/sh ./mysqld_safe
mysql 3578 3441 0 10:23 pts/3 00:00:00
/usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql
--datadir=/usr/local/mysql/var --user=mysql
--log-error=/usr/local/mysql/var/xen-server.err
--pid-file=/usr/local/mysql/var/xen-server.pid
--socket=/tmp/mysql.sock --port=3306
root      3616  3258  0 10:27 pts/3    00:00:00 grep mysqld
```

注意进程号为3578的进程，该进程就是MySQL实例。在上述例子中使用了mysqld_safe命令来启动数据库，当然启动MySQL实例的方法还有很多，在各种平台下的方式可能又会有所不同。在这里不一一赘述。

当启动实例时，MySQL数据库会去读取配置文件，根据配置文件的参数来启动数据库实例。这与Oracle的参数文件（spfile）相似，不同的是，Oracle中如果没有参数文件，在启动实例时会提示找不到该参数文件，数据库启动失败。而在MySQL数据库中，可以没有配置文件，在这种情况下，MySQL会按照编译时的默认参数设置启动实例。用以下命令可以查看当MySQL数据库实例启动时，会在哪些位置查找配置文件。

```
[root@xen-server bin]# mysql --help | grep my.cnf
order of preference, my.cnf, $MYSQL_TCP_PORT,
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf ~/.my.cnf
```

可以看到，MySQL数据库是按/etc/my.cnf→/etc/mysql/my.cnf→/usr/local/mysql/etc/my.cnf→～/.my.cnf的顺序读取配置文件的。可能有读者会问：“如果几个配置文件中都有同一个参数，MySQL数据库以哪个配置文件为准？”答案很简单，MySQL数据库会以读取到的最后一个配置文件中的参数为准。在Linux环境下，配置文件一般放在/etc/my.cnf下。在Windows平台下，配置文件的后缀名可能是.cnf，也可能是.ini。例如在Windows操作系统下运行mysql--help，可以找到如下类似内容：

```
    Default options are read from the following files in the given order:
    C:\Windows\my.ini C:\Windows\my.cnf C:\my.ini C:\my.cnf C:\Program Files\
MySQL\M
    \MySQL Server 5.1\my.cnf
```

配置文件中有一个参数datadir，该参数指定了数据库所在的路径。在Linux操作系统下默认datadir为/usr/local/mysql/data，用户可以修改该参数，当然也可以使用该路径，不过该路径只是一个链接，具体如下：

```
mysql>SHOW VARIABLES LIKE 'datadir'\G;
*************************** 1. row ***************************
Variable_name: datadir
        Value: /usr/local/mysql/data/
1 row in set (0.00 sec)1 row in set (0.00 sec)

mysql>system ls-lh /usr/local/mysql/data
total 32K
drwxr-xr-x  2 root mysql 4.0K Aug  6 16:23 bin
drwxr-xr-x  2 root mysql 4.0K Aug  6 16:23 docs
drwxr-xr-x  3 root mysql 4.0K Aug  6 16:04 include
drwxr-xr-x  3 root mysql 4.0K Aug  6 16:04 lib
drwxr-xr-x  2 root mysql 4.0K Aug  6 16:23 libexec
drwxr-xr-x 10 root mysql 4.0K Aug  6 16:23 mysql-test
drwxr-xr-x  5 root mysql 4.0K Aug  6 16:04 share
drwxr-xr-x  5 root mysql 4.0K Aug  6 16:23 sql-bench
lrwxrwxrwx  1 root mysql   16 Aug  6 16:05 data -> /opt/mysql_data/
```

从上面可以看到，其实 data 目录是一个链接，该链接指向了 /opt/mysql_data 目录。当然，用户必须保证 /opt/mysql_data 的用户和权限，使得只有 mysql 用户和组可以访问（通常 MySQL 数据库的权限为 mysql：mysql）。

1.2 MySQL 体系结构

由于工作的缘故，笔者的大部分时间需要与开发人员进行数据库方面的沟通，并对他们进行培训。不论他们是 DBA，还是开发人员，似乎都对 MySQL 的体系结构了解得不够透彻。很多人喜欢把 MySQL 与他们以前使用的 SQL Server、Oracle、DB2 作比较。因此笔者常常会听到这样的疑问：

- ❑ 为什么 MySQL 不支持全文索引？
- ❑ MySQL 速度快是因为它不支持事务吗？
- ❑ 数据量大于 1000 万时 MySQL 的性能会急剧下降吗？

……

对于 MySQL 数据库的疑问有很多很多，在解释这些问题之前，笔者认为不管对于使用哪种数据库的开发人员，了解数据库的体系结构都是最为重要的内容。

在给出体系结构图之前，用户应该理解了前一节提出的两个概念：数据库和数据库实例。很多人会把这两个概念混淆，即 MySQL 是数据库，MySQL 也是数据库实例。

这样来理解 Oracle 和 Microsoft SQL Server 数据库可能是正确的，但是这会给以后理解 MySQL 体系结构中的存储引擎带来问题。从概念上来说，数据库是文件的集合，是依照某种数据模型组织起来并存放于二级存储器中的数据集合；数据库实例是程序，是位于用户与操作系统之间的一层数据管理软件，用户对数据库数据的任何操作，包括数据库定义、数据查询、数据维护、数据库运行控制等都是在数据库实例下进行的，应用程序只有通过数据库实例才能和数据库打交道。

如果这样讲解后读者还是不明白，那这里再换一种更为直白的方式来解释：数据库是由一个个文件组成（一般来说都是二进制的文件）的，要对这些文件执行诸如 SELECT、INSERT、UPDATE 和 DELETE 之类的数据库操作是不能通过简单的操作文件来更改数据库的内容，需要通过数据库实例来完成对数据库的操作。所以，用户把 Oracle、SQL Server、MySQL 简单地理解成数据库可能是有失偏颇的，虽然在实际使用中并不会这么强调两者之间的区别。

好了，在给出上述这些复杂枯燥的定义后，现在可以来看看 MySQL 数据库的体系结构了，其结构如图 1-1 所示（摘自 MySQL 官方手册）。

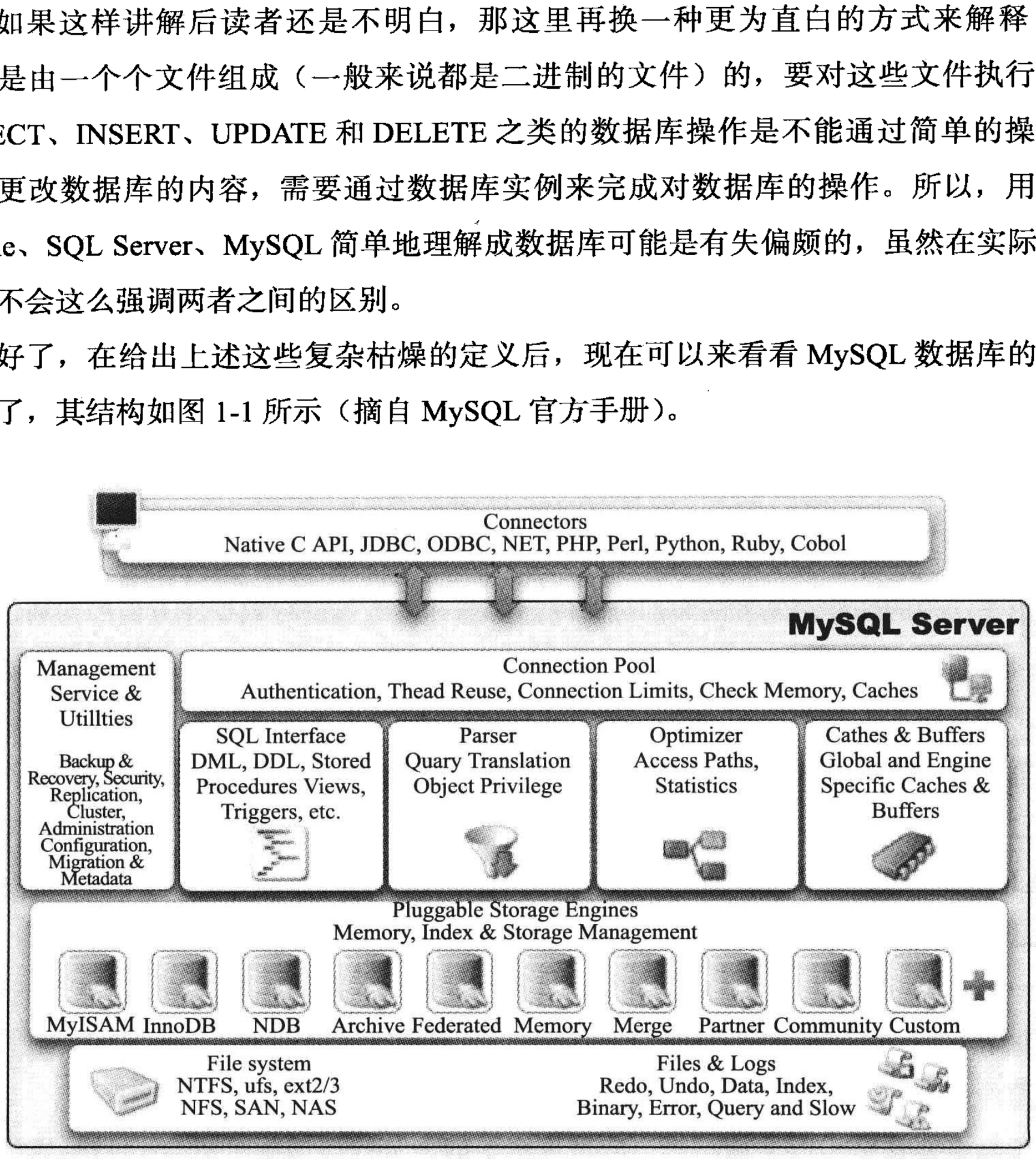


图 1-1　MySQL 体系结构

从图 1-1 可以发现，MySQL 由以下几部分组成：

- 连接池组件
- 管理服务和工具组件
- SQL 接口组件
- 查询分析器组件
- 优化器组件
- 缓冲（Cache）组件
- 插件式存储引擎
- 物理文件

从图 1-1 还可以发现，MySQL 数据库区别于其他数据库的最重要的一个特点就是其插件式的表存储引擎。MySQL 插件式的存储引擎架构提供了一系列标准的管理和服务支持，这些标准与存储引擎本身无关，可能是每个数据库系统本身都必需的，如 SQL 分析器和优化器等，而存储引擎是底层物理结构的实现，每个存储引擎开发者可以按照自己的意愿来进行开发。

需要特别注意的是，存储引擎是基于表的，而不是数据库。此外，要牢记图 1-1 的 MySQL 体系结构，它对于以后深入理解 MySQL 数据库会有极大的帮助。

1.3 MySQL 存储引擎

通过 1.2 节大致了解了 MySQL 数据库独有的插件式体系结构，并了解到存储引擎是 MySQL 区别于其他数据库的一个最重要特性。存储引擎的好处是，每个存储引擎都有各自的特点，能够根据具体的应用建立不同存储引擎表。对于开发人员来说，存储引擎对其是透明的，但了解各种存储引擎的区别对于开发人员来说也是有好处的。对于 DBA 来说，他们应该深刻地认识到 MySQL 数据库的核心在于存储引擎。

由于 MySQL 数据库的开源特性，用户可以根据 MySQL 预定义的存储引擎接口编写自己的存储引擎。若用户对某一种存储引擎的性能或功能不满意，可以通过修改源码来得到想要的特性，这就是开源带给我们的方便与力量。比如，eBay 的工程师 Igor Chernyshev 对 MySQL Memory 存储引擎的改进（http://code.google.com/p/mysql-heap-dynamic-rows/）并应用于 eBay 的 Personalization Platform，类似的修改还有 Google 和 Facebook 等公司。笔者曾尝试过对 InnoDB 存储引擎的缓冲池进行扩展，为其添加了基

于SSD的辅助缓冲池[⊖]，通过利用SSD的高随机读取性能来进一步提高数据库本身的性能。当然，MySQL数据库自身提供的存储引擎已经足够满足绝大多数应用的需求。如果用户有兴趣，完全可以开发自己的存储引擎，满足自己特定的需求。MySQL官方手册的第16章给出了编写自定义存储引擎的过程，不过这已超出了本书所涵盖的范围。

由于MySQL数据库开源特性，存储引擎可以分为MySQL官方存储引擎和第三方存储引擎。有些第三方存储引擎很强大，如大名鼎鼎的InnoDB存储引擎（最早是第三方存储引擎，后被Oracle收购），其应用就极其广泛，甚至是MySQL数据库OLTP（Online Transaction Processing在线事务处理）应用中使用最广泛的存储引擎。还是那句话，用户应该根据具体的应用选择适合的存储引擎，以下是对一些存储引擎的简单介绍，以便于读者选择存储引擎时参考。

1.3.1 InnoDB存储引擎

InnoDB存储引擎支持事务，其设计目标主要面向在线事务处理（OLTP）的应用。其特点是行锁设计、支持外键，并支持类似于Oracle的非锁定读，即默认读取操作不会产生锁。从MySQL数据库5.5.8版本开始，InnoDB存储引擎是默认的存储引擎。

InnoDB存储引擎将数据放在一个逻辑的表空间中，这个表空间就像黑盒一样由InnoDB存储引擎自身进行管理。从MySQL 4.1（包括4.1）版本开始，它可以将每个InnoDB存储引擎的表单独存放到一个独立的ibd文件中。此外，InnoDB存储引擎支持用裸设备（row disk）用来建立其表空间。

InnoDB通过使用多版本并发控制（MVCC）来获得高并发性，并且实现了SQL标准的4种隔离级别，默认为REPEATABLE级别。同时，使用一种被称为next-key locking的策略来避免幻读（phantom）现象的产生。除此之外，InnoDB储存引擎还提供了插入缓冲（insert buffer）、二次写（double write）、自适应哈希索引（adaptive hash index）、预读（read ahead）等高性能和高可用的功能。

对于表中数据的存储，InnoDB存储引擎采用了聚集（clustered）的方式，因此每张表的存储都是按主键的顺序进行存放。如果没有显式地在表定义时指定主键，InnoDB存储引擎会为每一行生成一个6字节的ROWID，并以此作为主键。

⊖ 详见：http://code.google.com/p/david-mysql-tools/wiki/innodb_secondary_buffer_pool

InnoDB 存储引擎是 MySQL 数据库最为常用的一种引擎，而 Facebook、Google、Yahoo！等公司的成功应用已经证明了 InnoDB 存储引擎具备的高可用性、高性能以及高可扩展性。

1.3.2 MyISAM 存储引擎

MyISAM 存储引擎不支持事务、表锁设计，支持全文索引，主要面向一些 OLAP 数据库应用。在 MySQL 5.5.8 版本之前 MyISAM 存储引擎是默认的存储引擎（除 Windows 版本外）。数据库系统与文件系统很大的一个不同之处在于对事务的支持，然而 MyISAM 存储引擎是不支持事务的。究其根本，这也不是很难理解。试想用户是否在所有的应用中都需要事务呢？在数据仓库中，如果没有 ETL 这些操作，只是简单的报表查询是否还需要事务的支持呢？此外，MyISAM 存储引擎的另一个与众不同的地方是它的缓冲池只缓存（cache）索引文件，而不缓冲数据文件，这点和大多数的数据库都非常不同。

MyISAM 存储引擎表由 MYD 和 MYI 组成，MYD 用来存放数据文件，MYI 用来存放索引文件。可以通过使用 myisampack 工具来进一步压缩数据文件，因为 myisampack 工具使用赫夫曼（Huffman）编码静态算法来压缩数据，因此使用 myisampack 工具压缩后的表是只读的，当然用户也可以通过 myisampack 来解压数据文件。

在 MySQL 5.0 版本之前，MyISAM 默认支持的表大小为 4GB，如果需要支持大于 4GB 的 MyISAM 表时，则需要制定 MAX_ROWS 和 AVG_ROW_LENGTH 属性。从 MySQL 5.0 版本开始，MyISAM 默认支持 256TB 的单表数据，这足够满足一般应用需求。

注意 对于 MyISAM 存储引擎表，MySQL 数据库只缓存其索引文件，数据文件的缓存交由操作系统本身来完成，这与其他使用 LRU 算法缓存数据的大部分数据库大不相同。此外，在 MySQL 5.1.23 版本之前，无论是在 32 位还是 64 位操作系统环境下，缓存索引的缓冲区最大只能设置为 4GB。在之后的版本中，64 位系统可以支持大于 4GB 的索引缓冲区。

1.3.3 NDB 存储引擎

2003 年，MySQL AB 公司从 Sony Ericsson 公司收购了 NDB 集群引擎（见图 1-1）。

NDB存储引擎是一个集群存储引擎，类似于Oracle的RAC集群，不过与Oracle RAC share everything架构不同的是，其结构是share nothing的集群架构，因此能提供更高的可用性。NDB的特点是数据全部放在内存中（从MySQL 5.1版本开始，可以将非索引数据放在磁盘上），因此主键查找（primary key lookups）的速度极快，并且通过添加NDB数据存储节点（Data Node）可以线性地提高数据库性能，是高可用、高性能的集群系统。

关于NDB存储引擎，有一个问题值得注意，那就是NDB存储引擎的连接操作（JOIN）是在MySQL数据库层完成的，而不是在存储引擎层完成的。这意味着，复杂的连接操作需要巨大的网络开销，因此查询速度很慢。如果解决了这个问题，NDB存储引擎的市场应该是非常巨大的。

注意 MySQL NDB Cluster存储引擎有社区版本和企业版本两种，并且NDB Cluster已作为Carrier Grade Edition单独下载版本而存在，可以通过http://dev.mysql.com/downloads/cluster/index.html获得最新版本的NDB Cluster存储引擎。

1.3.4 Memory存储引擎

Memory存储引擎（之前称HEAP存储引擎）将表中的数据存放在内存中，如果数据库重启或发生崩溃，表中的数据都将消失。它非常适合用于存储临时数据的临时表，以及数据仓库中的纬度表。Memory存储引擎默认使用哈希索引，而不是我们熟悉的B+树索引。

虽然Memory存储引擎速度非常快，但在使用上还是有一定的限制。比如，只支持表锁，并发性能较差，并且不支持TEXT和BLOB列类型。最重要的是，存储变长字段（varchar）时是按照定常字段（char）的方式进行的，因此会浪费内存（这个问题之前已经提到，eBay的工程师Igor Chernyshev已经给出了patch解决方案）。

此外有一点容易被忽视，MySQL数据库使用Memory存储引擎作为临时表来存放查询的中间结果集（intermediate result）。如果中间结果集大于Memory存储引擎表的容量设置，又或者中间结果含有TEXT或BLOB列类型字段，则MySQL数据库会把其转换到MyISAM存储引擎表而存放到磁盘中。之前提到MyISAM不缓存数据文件，因此这时产生的临时表的性能对于查询会有损失。

1.3.5 Archive 存储引擎

Archive 存储引擎只支持 INSERT 和 SELECT 操作，从 MySQL 5.1 开始支持索引。Archive 存储引擎使用 zlib 算法将数据行（row）进行压缩后存储，压缩比一般可达 1：10。正如其名字所示，Archive 存储引擎非常适合存储归档数据，如日志信息。Archive 存储引擎使用行锁来实现高并发的插入操作，但是其本身并不是事务安全的存储引擎，其设计目标主要是提供高速的插入和压缩功能。

1.3.6 Federated 存储引擎

Federated 存储引擎表并不存放数据，它只是指向一台远程 MySQL 数据库服务器上的表。这非常类似于 SQL Server 的链接服务器和 Oracle 的透明网关，不同的是，当前 Federated 存储引擎只支持 MySQL 数据库表，不支持异构数据库表。

1.3.7 Maria 存储引擎

Maria 存储引擎是新开发的引擎，设计目标主要是用来取代原有的 MyISAM 存储引擎，从而成为 MySQL 的默认存储引擎。Maria 存储引擎的开发者是 MySQL 的创始人之一的 Michael Widenius。因此，它可以看做是 MyISAM 的后续版本。Maria 存储引擎的特点是：支持缓存数据和索引文件，应用了行锁设计，提供了 MVCC 功能，支持事务和非事务安全的选项，以及更好的 BLOB 字符类型的处理性能。

1.3.8 其他存储引擎

除了上面提到的 7 种存储引擎外，MySQL 数据库还有很多其他的存储引擎，包括 Merge、CSV、Sphinx 和 Infobright，它们都有各自使用的场合，这里不再一一介绍。在了解 MySQL 数据库拥有这么多存储引擎后，现在我可以回答 1.2 节中提到的问题了。

- 为什么 MySQL 数据库不支持全文索引？不！ MySQL 支持，MyISAM、InnoDB（1.2 版本）和 Sphinx 存储引擎都支持全文索引。
- MySQL 数据库速度快是因为不支持事务？错！虽然 MySQL 的 MyISAM 存储引擎不支持事务，但是 InnoDB 支持。“快”是相对于不同应用来说的，对于 ETL 这种操作，MyISAM 会有其优势，但在 OLTP 环境中，InnoDB 存储引擎的效率

更好。

- 当表的数据量大于 1000 万时 MySQL 的性能会急剧下降吗？不！ MySQL 是数据库，不是文件，随着数据行数的增加，性能当然会有所下降，但是这些下降不是线性的，如果用户选择了正确的存储引擎，以及正确的配置，再多的数据量 MySQL 也能承受。如官方手册上提及的，Mytrix 和 Inc. 在 InnoDB 上存储超过 1 TB 的数据，还有一些其他网站使用 InnoDB 存储引擎，处理插入 / 更新的操作平均 800 次 / 秒。

1.4　各存储引擎之间的比较

通过 1.3 节的介绍，我们了解了存储引擎是 MySQL 体系结构的核心。本节我们将通过简单比较几个存储引擎来让读者更直观地理解存储引擎的概念。图 1-2 取自于 MySQL 的官方手册，展现了一些常用 MySQL 存储引擎之间的不同之处，包括存储容量的限制、事务支持、锁的粒度、MVCC 支持、支持的索引、备份和复制等。

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✔		✔		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✔	✔	✔
Geospatial support	✔					
B-Tree indexes	✔	✔	✔	✔		✔
Hash indexes			✔	✔		✔
Full text search index	✔					
Clustered index				✔		
Data Caches			✔	✔		✔
Index Caches	✔		✔	✔		✔
Compressed data	✔				✔	
Encrypted data (via function)	✔	✔	✔	✔	✔	✔
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✔
Replication support	✔	✔	✔	✔	✔	✔
Foreign key support				✔		
Backup/Point-in-time recovery	✔	✔	✔	✔	✔	✔
Query cache support	✔	✔	✔	✔	✔	✔
Update Statistics for Data Dictionary	✔	✔	✔	✔	✔	✔

图 1-2　不同 MySQL 存储引擎相关特性比较

可以看到，每种存储引擎的实现都不相同。有些竟然不支持事务，相信在任何一本关于数据库原理的书中，可能都会提到数据库与传统文件系统的最大区别在于数据库是支持事务的。而 MySQL 数据库的设计者在开发时却认为可能不是所有的应用都需要事务，所以存在不支持事务的存储引擎。更有不明其理的人把 MySQL 称做文件系统数据库，其实不然，只是 MySQL 数据库的设计思想和存储引擎的关系可能让人产生了理解上的偏差。

可以通过 SHOW ENGINES 语句查看当前使用的 MySQL 数据库所支持的存储引擎，也可以通过查找 information_schema 架构下的 ENGINES 表，如下所示：

```
mysql>SHOW ENGINES\G;
*************************** 1. row ***************************
      Engine: InnoDB
     Support: YES
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MEMORY
     Support: YES
```

```
      Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: MyISAM
     Support: DEFAULT
     Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
          XA: NO
  Savepoints: NO
8 rows in set (0.00 sec)
```

下面将通过MySQL提供的示例数据库来简单显示各存储引擎之间的不同。这里将分别运行以下语句，然后统计每次使用各存储引擎后表的大小。

```
mysql>CREATE TABLE mytest Engine=MyISAM
    ->AS SELECT * FROM salaries;
Query OK, 2844047 rows affected (4.37 sec)
Records: 2844047  Duplicates: 0  Warnings: 0

mysql>ALTER TABLE mytest Engine=InnoDB;
Query OK, 2844047 rows affected (15.86 sec)
Records: 2844047  Duplicates: 0  Warnings: 0

mysql>ALTER TABLE mytest Engine=ARCHIVE;
Query OK, 2844047 rows affected (16.03 sec)
Records: 2844047  Duplicates: 0  Warnings: 0
```

通过每次的统计，可以发现当最初表使用 MyISAM 存储引擎时，表的大小为 40.7MB，使用 InnoDB 存储引擎时表增大到了 113.6MB，而使用 Archive 存储引擎时表的大小却只有 20.2MB。该例子只从表的大小方面简单地揭示了各存储引擎的不同。

注意 MySQL 提供了一个非常好的用来演示 MySQL 各项功能的示例数据库，如 SQL Server 提供的 AdventureWorks 示例数据库和 Oracle 提供的示例数据库。据我所知，知道 MySQL 示例数据库的人很少，可能是因为这个示例数据库没有在安装的时候提示用户是否安装（如 Oracle 和 SQL Server）以及这个示例数据库的下载竟然和文档放在一起。用户可以通过以下地址找到并下载示例数据库：http://dev.mysql.com/doc/。

1.5 连接 MySQL

本节将介绍连接 MySQL 数据库的常用方式。需要理解的是，连接 MySQL 操作是一个连接进程和 MySQL 数据库实例进行通信。从程序设计的角度来说，本质上是进程通信。如果对进程通信比较了解，可以知道常用的进程通信方式有管道、命名管道、命名字、TCP/IP 套接字、UNIX 域套接字。MySQL 数据库提供的连接方式从本质上看都是上述提及的进程通信方式。

1.5.1 TCP/IP

TCP/IP 套接字方式是 MySQL 数据库在任何平台下都提供的连接方式，也是网络中使用得最多的一种方式。这种方式在 TCP/IP 连接上建立一个基于网络的连接请求，一般情况下客户端（client）在一台服务器上，而 MySQL 实例（server）在另一台服务器上，这两台机器通过一个 TCP/IP 网络连接。例如用户可以在 Windows 服务器下请求一台远程 Linux 服务器下的 MySQL 实例，如下所示：

```
C:\>mysql -h192.168.0.101 -u david -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18358
```

```
Server version: 5.0.77-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help.Type '\c' to clear the current input statement.

mysql>
```

这里的客户端是Windows，它向一台Host IP为192.168.0.101的MySQL实例发起了TCP/IP连接请求，并且连接成功。之后就可以对MySQL数据库进行一些数据库操作，如DDL和DML等。

这里需要注意的是，在通过TCP/IP连接到MySQL实例时，MySQL数据库会先检查一张权限视图，用来判断发起请求的客户端IP是否允许连接到MySQL实例。该视图在mysql架构下，表名为user，如下所示：

```
mysql>USE mysql;
Database changed
mysql>SELECT host,user,password FROM user;
*************************** 1. row ***************************
host: 192.168.24.%
user: root
password: *75DBD4FA548120B54FE693006C41AA9A16DE8FBE
*************************** 2. row ***************************
host: nineyou0-43
user: root
password: *75DBD4FA548120B54FE693006C41AA9A16DE8FBE
*************************** 3. row ***************************
host: 127.0.0.1
user: root
password: *75DBD4FA548120B54FE693006C41AA9A16DE8FBE
*************************** 4. row ***************************
host: 192.168.0.100
user: zlm
password: *DAE0939275CC7CD8E0293812A31735DA9CF0953C
*************************** 5. row ***************************
host: %
user: david
password:
5 rows in set (0.00 sec)
```

从这张权限表中可以看到，MySQL允许david这个用户在任何IP段下连接该实例，并且不需要密码。此外，还给出了root用户在各个网段下的访问控制权限。

1.5.2 命名管道和共享内存

在 Windows 2000、Windows XP、Windows 2003 和 Windows Vista 以及在此之上的平台上，如果两个需要进程通信的进程在同一台服务器上，那么可以使用命名管道，Microsoft SQL Server 数据库默认安装后的本地连接也是使用命名管道。在 MySQL 数据库中须在配置文件中启用 --enable-named-pipe 选项。在 MySQL 4.1 之后的版本中，MySQL 还提供了共享内存的连接方式，这是通过在配置文件中添加 --shared-memory 实现的。如果想使用共享内存的方式，在连接时，MySQL 客户端还必须使用 --protocol=memory 选项。

1.5.3 UNIX 域套接字

在 Linux 和 UNIX 环境下，还可以使用 UNIX 域套接字。UNIX 域套接字其实不是一个网络协议，所以只能在 MySQL 客户端和数据库实例在一台服务器上的情况下使用。用户可以在配置文件中指定套接字文件的路径，如 --socket=/tmp/mysql.sock。当数据库实例启动后，用户可以通过下列命令来进行 UNIX 域套接字文件的查找：

```
mysql>SHOW VARIABLES LIKE 'socket';
*************************** 1. row ***************************
Variable_name: socket
        Value: /tmp/mysql.sock
1 row in set (0.00 sec)
```

在知道了 UNIX 域套接字文件的路径后，就可以使用该方式进行连接了，如下所示：

```
[root@stargazer ~]# mysql -udavid -S /tmp/mysql.sock
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20333
Server version: 5.0.77-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help.Type '\c' to clear the buffer.

mysql>
```

1.6 小结

本章首先介绍了数据库和数据库实例的定义，紧接着分析了 MySQL 数据库的体系

结构，从而进一步突出强调了“实例”和“数据库”的区别。相信不管是 MySQL DBA 还是 MySQL 的开发人员都应该从宏观上了解了 MySQL 体系结构，特别是 MySQL 独有的插件式存储引擎的概念。因为很多 MySQL 用户很少意识到这一点，这给他们的管理、使用和开发带来了困扰。

本章还详细讲解了各种常见的表存储引擎的特性、适用情况以及它们之间的区别，以便于大家在选择存储引擎时作为参考。最后强调一点，虽然 MySQL 有许多的存储引擎，但是它们之间不存在优劣性的差异，用户应根据不同的应用选择适合自己的存储引擎。当然，如果你能力很强，完全可以修改存储引擎的源代码，甚至是创建属于自己特定应用的存储引擎，这不就是开源的魅力吗？

第 2 章　InnoDB 存储引擎

InnoDB 是事务安全的 MySQL 存储引擎，设计上采用了类似于 Oracle 数据库的架构。通常来说，InnoDB 存储引擎是 OLTP 应用中核心表的首选存储引擎。同时，也正是因为 InnoDB 的存在，才使 MySQL 数据库变得更有魅力。本章将详细介绍 InnoDB 存储引擎的体系架构及其不同于其他存储引擎的特性。

2.1　InnoDB 存储引擎概述

InnoDB 存储引擎最早由 Innobase Oy 公司[⊖]开发，被包括在 MySQL 数据库所有的二进制发行版本中，从 MySQL 5.5 版本开始是默认的表存储引擎（之前的版本 InnoDB 存储引擎仅在 Windows 下为默认的存储引擎）。该存储引擎是第一个完整支持 ACID 事务的 MySQL 存储引擎（BDB 是第一个支持事务的 MySQL 存储引擎，现在已经停止开发），其特点是行锁设计、支持 MVCC、支持外键、提供一致性非锁定读，同时被设计用来最有效地利用以及使用内存和 CPU。

Heikki Tuuri（1964 年，芬兰赫尔辛基）是 InnoDB 存储引擎的创始人，和著名的 Linux 创始人 Linus 是芬兰赫尔辛基大学校友。在 1990 年获得赫尔辛基大学的数学逻辑博士学位后，他于 1995 年成立 Innobase Oy 公司并担任 CEO。同时，在 InnoDB 存储引擎的开发团队中，有来自中国科技大学的 Calvin Sun。而最近又有一个中国人 Jimmy Yang 也加入了 InnoDB 存储引擎的核心开发团队，负责全文索引的开发，其之前任职于 Sybase 数据库公司，负责数据库的相关开发工作。

InnoDB 存储引擎已经被许多大型网站使用，如用户熟知的 Google、Yahoo!、Facebook、YouTube、Flickr，在网络游戏领域有《魔兽世界》、《Second Life》、《神兵玄奇》等。我不是 MySQL 数据库的布道者，也不是 InnoDB 的鼓吹者，但是我认为当前实施一个新的 OLTP 项目不使用 MySQL InnoDB 存储引擎将是多么的愚蠢。

⊖ 2006年该公司已经被Oracle公司收购。

从MySQL数据库的官方手册可得知，著名的Internet新闻站点Slashdot.org运行在InnoDB上。Mytrix、Inc.在InnoDB上存储超过1 TB的数据，还有一些其他站点在InnoDB上处理插入/更新操作的速度平均为800次/秒。这些都证明了InnoDB是一个高性能、高可用、高可扩展的存储引擎。

InnoDB存储引擎同MySQL数据库一样，在GNU GPL 2下发行。更多有关MySQL证书的信息，可参考http://www.mysql.com/about/legal/，这里不再详细介绍。

2.2 InnoDB存储引擎的版本

InnoDB存储引擎被包含于所有MySQL数据库的二进制发行版本中。早期其版本随着MySQL数据库的更新而更新。从MySQL 5.1版本时，MySQL数据库允许存储引擎开发商以动态方式加载引擎，这样存储引擎的更新可以不受MySQL数据库版本的限制。所以在MySQL 5.1中，可以支持两个版本的InnoDB，一个是静态编译的InnoDB版本，可将其视为老版本的InnoDB；另一个是动态加载的InnoDB版本，官方称为InnoDB Plugin，可将其视为InnoDB 1.0.x版本。MySQL 5.5版本中又将InnoDB的版本升级到了1.1.x。而在最近的MySQL 5.6版本中InnoDB的版本也随着升级为1.2.x版本。表2-1显示了各个版本中InnoDB存储引擎的功能。

表2-1 InnoDB各版本功能对比

版　本	功　能
老版本InnoDB	支持ACID、行锁设计、MVCC
InnoDB 1.0.x	继承了上述版本所有功能，增加了compress和dynamic页格式
InnoDB 1.1.x	继承了上述版本所有功能，增加了Linux AIO、多回滚段
InnoDB 1.2.x	继承了上述版本所有功能，增加了全文索引支持、在线索引添加

在现实工作中我发现很多MySQL数据库还是停留在MySQL 5.1版本，并使用InnoDB Plugin。很多DBA错误地认为InnoDB Plugin和InnoDB 1.1版本之间是没有区别的。但从表2-1中还是可以发现，虽然都增加了对于compress和dynamic页的支持，但是InnoDB Plugin是不支持Linux Native AIO功能的。此外，由于不支持多回滚段，InnoDB Plugin支持的最大支持并发事务数量也被限制在1023。而且随着MySQL 5.5版本的发布，InnoDB Plugin也变成了一个历史产品。

2.3 InnoDB 体系架构

通过第 1 章读者已经了解了 MySQL 数据库的体系结构，现在可能想更深入地了解 InnoDB 存储引擎的架构。图 2-1 简单显示了 InnoDB 的存储引擎的体系架构，从图可见，InnoDB 存储引擎有多个内存块，可以认为这些内存块组成了一个大的内存池，负责如下工作：

- 维护所有进程 / 线程需要访问的多个内部数据结构。
- 缓存磁盘上的数据，方便快速地读取，同时在对磁盘文件的数据修改之前在这里缓存。
- 重做日志（redo log）缓冲。

……

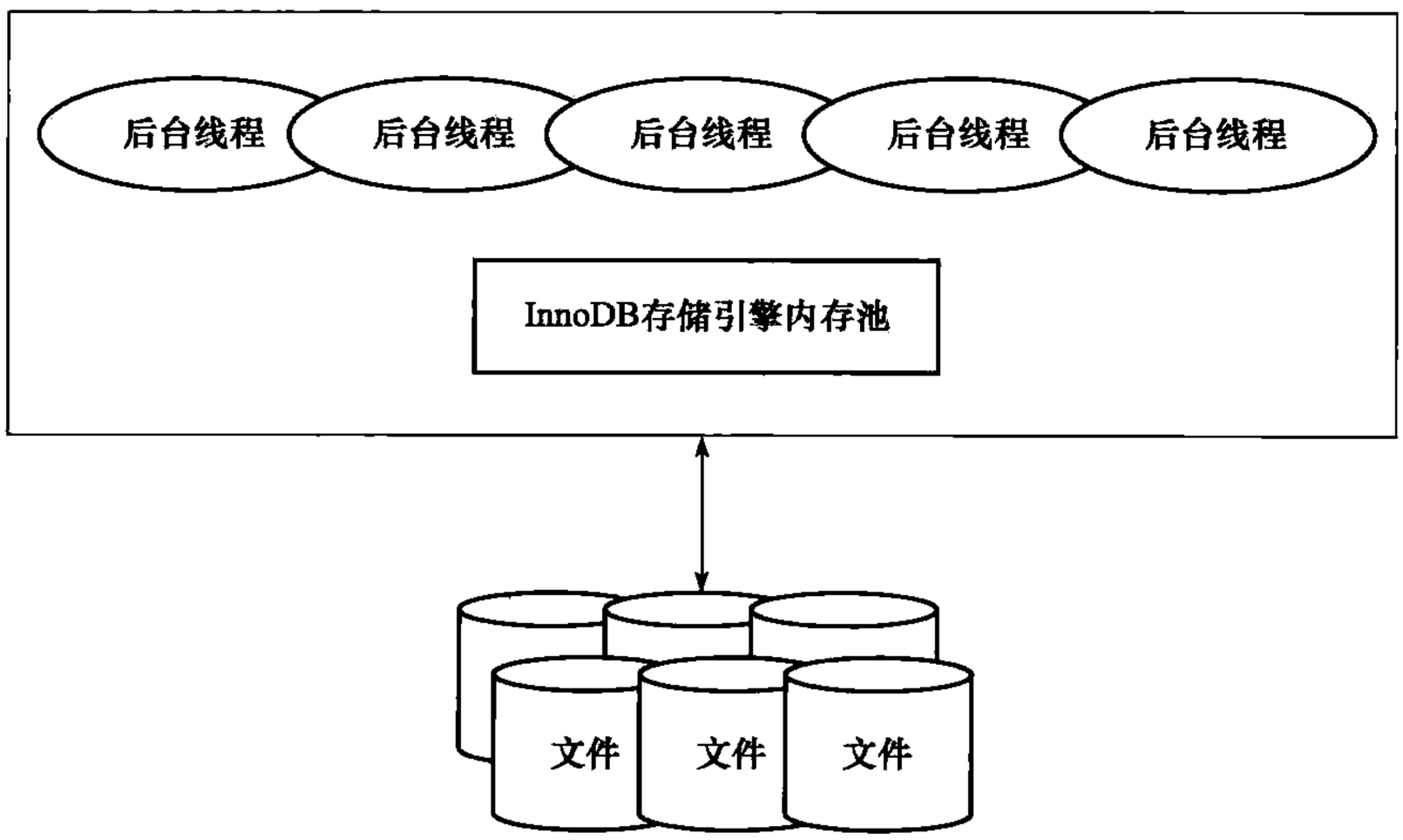


图 2-1 InnoDB 存储引擎体系架构

后台线程的主要作用是负责刷新内存池中的数据，保证缓冲池中的内存缓存的是最近的数据。此外将已修改的数据文件刷新到磁盘文件，同时保证在数据库发生异常的情况下 InnoDB 能恢复到正常运行状态。

2.3.1 后台线程

InnoDB 存储引擎是多线程的模型，因此其后台有多个不同的后台线程，负责处理不

同的任务。

1. Master Thread

Master Thread是一个非常核心的后台线程，主要负责将缓冲池中的数据异步刷新到磁盘，保证数据的一致性，包括脏页的刷新、合并插入缓冲（INSERT BUFFER）、UNDO页的回收等。2.5节会详细地介绍各个版本中Master Thread的工作方式。

2. IO Thread

在InnoDB存储引擎中大量使用了AIO（Async IO）来处理写IO请求，这样可以极大提高数据库的性能。而IO Thread的工作主要是负责这些IO请求的回调（call back）处理。InnoDB 1.0版本之前共有4个IO Thread，分别是write、read、insert buffer和log IO thread。在Linux平台下，IO Thread的数量不能进行调整，但是在Windows平台下可以通过参数innodb_file_io_threads来增大IO Thread。从InnoDB 1.0.x版本开始，read thread和write thread分别增大到了4个，并且不再使用innodb_file_io_threads参数，而是分别使用innodb_read_io_threads和innodb_write_io_threads参数进行设置，如：

```
mysql>SHOW VARIABLES LIKE 'innodb_version'\G;
*************************** 1. row ***************************
Variable_name: innodb_version
        Value: 1.0.6
1 row in set (0.00 sec)

mysql>SHOW VARIABLES LIKE 'innodb_%io_threads'\G;
*************************** 1. row ***************************
Variable_name: innodb_read_io_threads
        Value: 4
*************************** 2. row ***************************
Variable_name: innodb_write_io_threads
        Value: 4
2 rows in set (0.00 sec)
```

可以通过命令SHOW ENGINE INNODB STATUS来观察InnoDB中的IO Thread：

```
mysql>SHOW ENGINE INNODB STATUS\G;
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
100719 21:55:26 INNODB MONITOR OUTPUT
=====================================
```

```
Per second averages calculated from the last 36 seconds
......
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (read thread)
I/O thread 4 state: waiting for i/o request (read thread)
I/O thread 5 state: waiting for i/o request (read thread)
I/O thread 6 state: waiting for i/o request (write thread)
I/O thread 7 state: waiting for i/o request (write thread)
I/O thread 8 state: waiting for i/o request (write thread)
I/O thread 9 state: waiting for i/o request (write thread)
......
---------------------------
END OF INNODB MONITOR OUTPUT
============================


1 row in set (0.01 sec)
```

可以看到 IO Thread 0 为 insert buffer thread。IO Thread 1 为 log thread。之后就是根据参数 innodb_read_io_threads 及 innodb_write_io_threads 来设置的读写线程，并且读线程的 ID 总是小于写线程。

3. Purge Thread

事务被提交后，其所使用的 undolog 可能不再需要，因此需要 PurgeThread 来回收已经使用并分配的 undo 页。在 InnoDB 1.1 版本之前，purge 操作仅在 InnoDB 存储引擎的 Master Thread 中完成。而从 InnoDB 1.1 版本开始，purge 操作可以独立到单独的线程中进行，以此来减轻 Master Thread 的工作，从而提高 CPU 的使用率以及提升存储引擎的性能。用户可以在 MySQL 数据库的配置文件中添加如下命令来启用独立的 Purge Thread：

```
[mysqld]
innodb_purge_threads=1
```

在 InnoDB 1.1 版本中，即使将 innodb_purge_threads 设为大于 1，InnoDB 存储引擎启动时也会将其设为 1，并在错误文件中出现如下类似的提示：

```
120529 22:54:16 [Warning] option 'innodb-purge-threads': unsigned value 4 adjusted to 1
```

从InnoDB 1.2版本开始，InnoDB支持多个Purge Thread，这样做的目的是为了进一步加快undo页的回收。同时由于Purge Thread需要离散地读取undo页，这样也能更进一步利用磁盘的随机读取性能。如用户可以设置4个Purge Thread：

```
mysql> SELECT VERSION()\G;
*************************** 1. row ***************************
VERSION(): 5.6.6
1 row in set (0.00 sec)

mysql> SHOW VARIABLES LIKE 'innodb_purge_threads'\G;
*************************** 1. row ***************************
Variable_name: innodb_purge_threads
        Value: 4
1 row in set (0.00 sec)
```

4. Page Cleaner Thread

Page Cleaner Thread是在InnoDB 1.2.x版本中引入的。其作用是将之前版本中脏页的刷新操作都放入到单独的线程中来完成。而其目的是为了减轻原Master Thread的工作及对于用户查询线程的阻塞，进一步提高InnoDB存储引擎的性能。

2.3.2 内存

1. 缓冲池

InnoDB存储引擎是基于磁盘存储的，并将其中的记录按照页的方式进行管理。因此可将其视为基于磁盘的数据库系统（Disk-base Database）。在数据库系统中，由于CPU速度与磁盘速度之间的鸿沟，基于磁盘的数据库系统通常使用缓冲池技术来提高数据库的整体性能。

缓冲池简单来说就是一块内存区域，通过内存的速度来弥补磁盘速度较慢对数据库性能的影响。在数据库中进行读取页的操作，首先将从磁盘读到的页存放在缓冲池中，这个过程称为将页“FIX”在缓冲池中。下一次再读相同的页时，首先判断该页是否在缓冲池中。若在缓冲池中，称该页在缓冲池中被命中，直接读取该页。否则，读取磁盘上的页。

对于数据库中页的修改操作，则首先修改在缓冲池中的页，然后再以一定的频率刷新到磁盘上。这里需要注意的是，页从缓冲池刷新回磁盘的操作并不是在每次页发生更