大型互联网公司资深架构师全面、系统、深入地分享MySQL技术

涵盖MySQL基础、开发、优化、运维与架构的方方面面知识

**阿里巴巴、蚂蚁金服及Dromara等知名公司与机构的5位技术专家力荐**

# MySQL技术大全

## 开发、优化与运维实战

## （视频教学版）

**15小时（84段）教学视频 + 450个示例 + 37个综合案例**

冰河◎编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

MySQL技术大全：开发、优化与运维实战：视频教学版/冰河编著．—北京：机械工业出版社，2020.11

ISBN 978-7-111-66898-5

Ⅰ．①M…　Ⅱ．①冰…　Ⅲ．①SQL语言－程序设计　Ⅳ．①TP311.132.3

中国版本图书馆CIP数据核字（2020）第222096号

MySQL 技术大全：开发、优化与运维实战（视频教学版）

出版发行：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：陈佳媛　　责任校对：姚志娟

印　　刷：北京捷迅佳彩印刷有限公司　　版　　次：2021 年 1 月第 1 版第 1 次印刷

开　　本：186mm×240mm　1/16　　印　　张：47.5

书　　号：ISBN 978-7-111-66898-5　　定　　价：199.00 元

客服电话：（010）88361066　88379833　68326294　　投稿热线：（010）88379604

华章网站：www.hzbook.com　　读者信箱：hzit@hzbook.com

本书法律顾问：北京大成律师事务所　韩光/邹晓东

# 前言

## 为何要写这本书

MySQL 具有小巧、灵活和免费等特性，这使得它越来越多地被用于企业的实际开发中。特别是 MySQL 数据库的开源特性，更使它得到了广泛应用。程序员要想进入 MySQL 开发领域，除了需要有扎实的编程基础外，还需要掌握 SQL 语句的编写，熟悉 MySQL 数据库的优化和运维，了解 MySQL 数据库的常见故障和解决方案，这样才能在竞争日益激烈的数据库领域提高竞争力，进而实现自身的价值。

目前，市面上介绍 MySQL 数据库技术的图书不少，但是真正从实战出发，全面介绍 MySQL 基础、环境搭建、开发、优化、运维和架构的图书却很少。本书结合大量的实战案例，详细地介绍掌握 MySQL 数据库所需要的各项技能，尤其是环境搭建、MySQL 优化、MySQL 运维和架构等相关内容。通过阅读本书，读者能够更加全面、深入、透彻地理解 MySQL 数据库技术，对书中所述内容稍加修改便可直接应用于自己的工作之中，从而提高自身的 MySQL 开发水平和项目实战能力。

## 本书特色

### 1．提供近15小时配套教学视频

为了让读者更加高效、直观地学习和理解本书中的重点内容和难点内容，笔者专门录制了近 15 小时（共 84 段）的配套教学视频辅助读者学习，相信读者结合教学视频，可以取得更好的学习效果。

### 2．内容非常全面，涵盖MySQL的所有重要技术点

本书非常全面地介绍了 MySQL 数据库的各个知识点，涵盖 MySQL 基础知识、环境搭建、开发、优化、运维和架构等。通过阅读本书，读者能够全面掌握 MySQL 数据库的各项技术要点。

### 3．给出大量的图解和实战案例

本书在介绍理论知识时都配有对应的图表，而且在知识点讲解后都给出了大量的实战案例，帮助读者更加直观地理解所学内容。读者只要很好地理解各个知识点并亲自动手运行每

个示例的源代码，就能够更加深入地理解和使用 MySQL。

#### 4．案例典型，实用性强

本书中的实战案例涉及 MySQL 技术的方方面面，都非常典型，而且这些案例具有很强的实用性，略加修改就可以迁移到自己的工作中，读者只要很好地理解和掌握这些案例，就能触类旁通，举一反三。

#### 5．讲解通俗易懂

本书从始至终都用通俗易懂的语言进行讲解，书中对每个概念都给出了清晰的定义，对每个知识点都给出了简明扼要的讲解，对每个案例都给出了清晰明了的实现步骤，而且讲解时言简意赅，这大大提升了读者的阅读体验。

## 本书知识体系

#### 第1篇　MySQL基础

本篇涵盖第 1～3 章，主要介绍数据库的定义与发展，以及数据库技术、MySQL 数据库的三大范式及存储引擎。

#### 第2篇　环境搭建

本篇涵盖第 4～6 章，主要介绍如何安装 VMware 虚拟机，如何安装 Windows、Mac OS X 和 CentOS 操作系统，以及如何在三大操作系统上安装和配置 MySQL 环境。

#### 第3篇　MySQL开发

本篇涵盖第 7～19 章，主要介绍 MySQL 的技术要点，包括如何操作数据库和数据表，MySQL 中的数据类型、运算符、函数、数据变更和数据查询，MySQL 中的索引、视图、触发器、存储过程和函数，以及 MySQL 分区、公用表表达式与生成列。

#### 第4篇　MySQL优化

本篇涵盖第 20～26 章，主要介绍 MySQL 中的查询优化、索引优化、SQL 语句优化、数据库优化、服务器优化、应用程序优化及其他优化技术等。

#### 第5篇　MySQL运维

本篇涵盖第 27～30 章，主要介绍 MySQL 中各种命令行工具的使用，各种日志的开启、查看、删除与关闭，数据的备份与恢复，以及 MySQL 中的账户管理等。

第6篇　MySQL架构

本篇涵盖第 31～33 章，主要介绍如何实现 MySQL 中的复制，如何搭建 MySQL 的读写分离环境，以及如何实现 MySQL 的高可用性等。

## 配书资源获取

本书附赠如下配书资源：

- 近 15 小时（共 84 段）配套教学视频；
- 所有案例的 SQL 源码文件。

这些配套资源需要读者自行下载。请在华章公司网站（www.hzbook.com）上搜索到本书，然后单击“资料下载”按钮，即可在本书页面上找到下载链接。

## 本书读者对象

- 想全面学习 MySQL 技术的人员；
- 想转行从事数据库开发的人员；
- 数据库管理人员；
- 数据库运维工程师；
- 希望提高数据库实战水平的人员；
- 数据库开发经理；
- 数据库架构师；
- 相关院校的学生；
- 专业培训机构的学员；
- 需要时常查阅 MySQL 技术和开发案例的人员。

## 阅读建议

- 不具备 MySQL 基础知识的读者，建议从第 1 章顺次阅读，并按照书中的操作步骤实现每一个案例。
- 有一定 MySQL 开发基础的读者，可以根据自身的实际情况有选择地阅读。
- 书中的每个案例，先自行思考如何实现，再阅读笔者介绍的方法，可以达到事半功倍的效果。
- 先理解书中介绍的相关技术原理，再亲自动手实现一遍书中的案例，理解会更加深刻。

## 勘误与支持

尽管笔者对技术有近乎完美的追求，但是由于 MySQL 体系庞大，所涉及的知识点众多，一本书很难讲解清楚所有的知识点。如果笔者有疏漏，恳请读者朋友能够及时批评和指正。如果您对本书有好的建议或者想法，可以通过以下方式进行反馈。

邮箱：hzbook2017@163.com 或 1028386804@qq.com

微信：sun_shine_lyz

微信公众号：冰河技术

## 致谢

感谢季敏（阿里巴巴技术专家，Seata 开源项目发起人）、肖宇（开源组织 Dromara 创始人，Soul 网关与 Hmily 分布式事务框架作者）、刘遄（LinuxProbe 网站创始人，RHCA 架构师，运维专家，畅销书《Linux 就该这么学》作者）、芋艿（芋道源码作者）和黄小邪（蚂蚁金服高级开发工程师）对我写作本书的大力支持和帮助！

感谢我的团队成员和许多一起交流过乃至合作过的朋友们！

感谢我的 CSDN 博客粉丝以及那些在我博客和公众号上留言的朋友们！

感谢我的家人，他们都以自己的方式在我写作期间默默地给予支持与鼓励！

感谢出版社参与本书出版的各位编辑，没有你们的辛勤工作和一丝不苟的精神，就不会有本书的高质量出版。

感谢其他支持、鼓励和帮助过我的人！

最后感谢本书读者，是你们的鞭策，才让我有动力完成写作本书的“艰巨”任务！

冰河

# 目录

# 第 2 篇 环境搭建

# 第 3 篇 MySQL 开发

# 第 4 篇　MySQL 优化

# 第 5 篇　MySQL 运维

# 第 6 篇 MySQL 架构

# 第 1 篇
# MySQL 基础

# 第 1 章　数据库概述

随着计算机技术的发展，数据在计算机上的存储形式也不断发生着变化。而数据库的诞生和发展经历了漫长的演化过程，每项存储技术的诞生都伴随着新的技术突破。本章就对数据库的基础知识做简单的介绍。

本章涉及的知识点有：

- 数据库的定义，了解数据库的基本概念。
- 数据库的发展，了解数据库的发展阶段。

## 1.1　数据库的定义

在某种程度上，数据库代表着一种存储技术，并不局限于某种存储形式。一个简单的数据库可以将数据只存储在某台特定的计算机上，供某个特定的用户使用，而一个复杂的数据库可以将数据分散存储到多台计算机上，能够供成千上万的用户同时使用。从存储容量上来说，一个数据库的存储容量可以小到只能够存储几 KB 的数据，也可以大到存储 TB 甚至是 PB 级别的数据。

### 1.1.1　数据库

数据库（DataBase，DB）从本质上讲就是一个文件系统，它能够将数据有组织地集合在一起，按照一定的规则长期存储到计算机的磁盘中，并且能够供多个用户共享和使用，同时，用户能够对数据库中的数据进行插入、删除、修改和查询操作。

数据库将数据进行集中存储和管理，有效地分离了应用程序和业务数据，降低了应用程序和业务数据之间的耦合性，大大简化了数据的存储和管理工作。同时，数据库提供了对存储数据的统一控制功能。

数据除了能够被存储在计算机的磁盘中，还能够被存储在计算机的内存中，所以在某种程度上，可以将数据库分为永久型数据库和内存型数据库。

### 1.1.2　数据库管理系统

数据库管理系统（DataBase Management System，DBMS）从本质上讲就是一个为管理数

据库中的数据而设计的一套管理系统。它依托数据库，对外提供统一管理数据库中数据的功能和接口，能够有效地对数据库的安全、认证、数据备份、数据恢复、数据传输等进行统一的管理。同时，数据库管理系统能够根据所依托的数据库模型对数据库进行相应的分类。大多数的数据库都是通过数据库管理系统对数据库中的数据进行管理和维护的。

### 1.1.3 数据表

对于关系型数据库来说，数据表是以一个二维数组的形式来存储和管理数据的，它能够存储和管理数据并操作数据的逻辑结构。通常，一个数据表由行和列组成，一行数据能够表示一条完整的基础信息，所以行在关系型数据库中是组织数据的基本单位；列也被称为字段，它能够表示行的一个属性，同时，每一列都有相应的数据类型和数据长度的定义。

例如，一个简单的商品信息表，每行数据都包含商品编号、商品名称、商品类型、商品价格和上架时间等，它们被称为列，也叫作字段，表示商品信息表一行所记录的一个属性，它们都有相应的数据类型和数据长度的定义。所有的列组成商品信息表中一条完整的行记录。

商品信息表中每行记录的定义如表 1-1 所示。

表 1-1　商品信息表中每行记录的定义表示

| 字 段 名 称 | 数 据 类 型 | 是否可为NULL | 是 否 主 键 | 默　认　值 |
|---|---|---|---|---|
| 商品编号 | varchar(32) | 否 | 是 | 空字符串 |
| 商品名称 | varchar(50) | 否 | 否 | 空字符串 |
| 商品类型 | varchar(30) | 否 | 否 | 空字符串 |
| 商品价格 | decimal(10, 2) | 否 | 否 | 0.00 |
| 上架时间 | datetime | 是 | 否 | NULL |

商品信息表中每行记录的数据可以表示为表 1-2 所示。

表 1-2　商品信息表中每行记录的数据表示

| 商 品 编 号 | 商 品 名 称 | 商 品 类 型 | 商 品 价 格 | 上 架 时 间 |
|---|---|---|---|---|
| 1000000001 | 破洞牛仔裤 | 女装/女士精品 | 79.90 | 2020-12-10 00:00:00 |
| 1000000002 | T恤 | 男装 | 49.90 | 2020-12-10 00:00:00 |
| 1000000003 | 苹果 | 水果 | 19.90 | 2020-12-11 00:00:00 |
| 1000000004 | 晾衣竿 | 居家用品 | 39.90 | 2020-12-15 00:00:00 |

### 1.1.4 数据类型

关系型数据库中的数据类型表示数据在数据库中的存储格式，其反映了数据在计算机中的存储格式。计算机根据不同的数据类型来组织和存储数据，并为不同数据类型的数据分配

不同的存储空间。

数据类型在大的分类上可以分为数值类型、日期和时间类型、字符串类型。

在关系型数据库中，表中的每个字段都会被指定一种数据类型。例如，表 1-1 中，将商品编号、商品名称和商品类型定义为字符串类型，将商品价格定义为数值类型（定点数类型），将上架时间定义为日期和时间类型。

### 1.1.5 运算符

运算符是一种运算符号，用来表示某种数据关系。运算符可以分为算术运算符、比较运算符、逻辑运算符和位运算符等。

### 1.1.6 函数

数据库中内置了一些函数，能够很方便地对数据进行数学计算、字符串处理、加密/解密及聚合处理等。相应地，函数可以分为数学函数、字符串函数、日期和时间函数、流程处理函数、加密与解密函数、数据聚合函数、获取数据库信息函数以及数据库中的其他函数等。

### 1.1.7 主键

在关系型数据库中，主键（Primary Key）又称为主码，能够唯一标识数据表中的一行记录。主键可以包含数据表中的一列或者多列，主键不能为空。同时，在同一个数据表中，主键列上不能有两行甚至多行相同的值，也就是说，在同一个数据表中，每行数据对应的主键列的值必须唯一。

例如，在表 1-1 中，将商品编号定义为商品信息表的主键，此时，当商品编号为空，或者商品编号在商品信息表中出现相同的值，则数据库会提示错误信息，查询不到相应的数据；如果将商品名称作为主键，则根据作为主键列的要求，商品名称不能重复，这与实际情况不符，所以商品名称字段不适合作为主键。

### 1.1.8 外键

外键从本质上讲就是一个引用，它引用的是另外一张表中的一列或者多列数据，被引用的表中的列需要具备主键约束或者唯一性约束。也就是说，被引用的列在其对应的数据表中能够唯一标识一行数据。外键反映的是两个表之间的连接关系。

例如，两个数据表分别为部门表和员工信息表。其中，部门表中包含两个字段，分别为部门编号和部门名称；员工信息表中包含员工编号、员工姓名、员工性别、员工生日、部门编号和入职日期等字段。

部门表中每行记录的定义如表 1-3 所示。

表 1-3　部门表中每行记录的定义表示

| 字 段 名 称 | 数 据 类 型 | 是否可为NULL | 是 否 主 键 | 默　认　值 |
|---|---|---|---|---|
| 部门编号 | varchar(32) | 否 | 是 | 空字符串 |
| 部门名称 | varchar(50) | 否 | 否 | 空字符串 |

员工信息表中每行记录的定义如表 1-4 所示。

表 1-4　员工信息表中每行记录的定义表示

| 字 段 名 称 | 数 据 类 型 | 是否可为NULL | 是 否 主 键 | 是 否 外 键 | 默　认　值 |
|---|---|---|---|---|---|
| 员工编号 | varchar(32) | 否 | 是 | 否 | 空字符串 |
| 员工姓名 | varchar(20) | 否 | 否 | 否 | 空字符串 |
| 员工性别 | char(2) | 否 | 否 | 否 | 未知 |
| 员工生日 | date | 是 | 否 | 否 | NULL |
| 部门编号 | varchar(32) | 否 | 否 | 是 | 空字符串 |
| 入职日期 | date | 否 | 否 | 否 | 当前日期 |

部门表中每行记录的数据如表 1-5 所示。

表 1-5　部门表中每行记录的数据表示

| 部 门 编 号 | 部 门 名 称 |
|---|---|
| 1000000001 | 技术部 |
| 1000000002 | 运营部 |
| 1000000003 | 市场部 |

员工信息表中每行记录的数据如表 1-6 所示。

表 1-6　员工信息表中每行记录的数据表示

| 员 工 编 号 | 员 工 姓 名 | 员 工 性 别 | 员 工 生 日 | 部 门 编 号 | 入 职 日 期 |
|---|---|---|---|---|---|
| 10000001 | 张三 | 男 | 1992-10-29 | 1000000001 | 2018-03-26 |
| 10000002 | 李四 | 男 | 1990-05-27 | 1000000001 | 2017-09-10 |
| 10000003 | 王五 | 女 | 1993-04-03 | 1000000002 | 2019-03-09 |
| 10000004 | 赵六 | 男 | 1995-06-20 | 1000000002 | 2017-04-19 |
| 10000005 | 杨七 | 女 | 1992-01-16 | 1000000003 | 2017-05-07 |
| 10000006 | 冯八 | 女 | 1994-03-26 | 1000000003 | 2020-10-16 |

可以看出，员工信息表中以部门编号为外键，而部门编号又是部门表中的主键，所以员工信息表中的外键引用的是部门表中的主键。

### 1.1.9 索引

索引从本质上来讲是一种单独的数据库结构，它能够单独地存储在计算机的磁盘上，包含着对相关数据表中所有数据的引用指针。通过索引能够快速定位并查询出数据表中的一行或者多行数据，而不必进行全表扫描。

在某种程度上，数据库的索引和书籍的目录有些类似。当查找书籍中的内容时，往往不会直接翻阅书籍的内容，这样查找起来相当烦琐；如果先根据书籍的目录定位到要查找的内容在书籍中的大概章节，然后再到相关的章节中去查找内容就比较简单了。

索引使查询能够快速到达计算机中的某个位置去搜寻数据文件，而不必对所有的数据进行扫描。索引的建立，可以大大提高数据查询的效率。

### 1.1.10 视图

视图从本质上来讲是数据库的一种虚表，它是由 SELECT 查询语句从一张表或者多张表中导出的一种虚表。不能向视图中插入、更新和删除数据，即视图不负责数据的实际存储。当通过视图修改数据时，实际上修改的是构成视图的基本表中的数据，当修改了构成视图的基本表中的数据时，视图中的数据也会随之改变。

使用视图能够大大简化数据库中表与表之间复杂的关联查询。

### 1.1.11 存储过程

存储过程是一种 SQL 语句集，经过编译后存储在数据库中，通过指定存储过程的名称和参数信息来调用存储过程，使其完成特定的功能。在创建存储过程的时候，可以自定义变量来存储一些中间结果的数据，也可以在存储过程中定义一些执行逻辑和执行流程。

存储过程经过一次编译后可以永久使用（只要不删除存储过程）。将一些复杂的查询逻辑封装在存储过程中重复使用，应用程序只需要调用存储过程的名称并传入相应的参数即可，大大简化了开发和数据查询的复杂度。另外，使用存储过程也可以防止用户直接访问数据表，只需要赋予用户对存储过程的访问权限即可。

### 1.1.12 触发器

触发器从本质上来讲是一种特殊的存储过程。触发器的执行不是由应用程序调用，也不是由手动执行的，而是由数据库中的事件执行的。当对某个表中的数据进行插入、更新和删除操作时，系统会自动执行相应的触发器。

在某种程度上，触发器和钩子函数有些类似。应用程序在执行某项操作时，会自动调用

相应的钩子函数，执行钩子函数的逻辑。而触发器是对数据表进行操作时自动执行的。

当对数据表中的数据执行插入、更新和删除操作，需要自动执行一些数据库逻辑时，可以使用触发器来实现。

### 1.1.13 存储引擎

存储引擎代表的是一种存储技术。对于每种存储引擎来说，其对应的存储机制、索引存放方式、索引技巧、数据库锁机制及数据的存储方式各不相同。

MySQL 中最常用到的存储引擎是 InnoDB 和 MyISAM。

## 1.2 数据库的发展

数据库技术经历了漫长的演化过程，在此期间各种存储技术层出不穷，在一定程度上促进了数据库技术的发展。数据库的发展经历了人工管理阶段、文件系统阶段、数据库系统阶段和云数据库阶段（笔者认为目前已经步入云数据库阶段）。

### 1.2.1 人工管理阶段

顾名思义，人工管理阶段需要人工管理和维护数据。此时，不会对数据进行保存操作，所有的数据都是由开发人员开发的程序进行管理，没有专门的系统或软件对数据进行存储、管理和维护。同时，数据无法在多个应用程序之间共享，也没有独立性。如果需要对数据的结构进行变更，就必须通过修改相应的应用程序来实现，复杂程度可想而知。

在人工管理阶段，程序开发人员需要付出大量额外的时间和精力来管理和维护数据，这无疑加重了程序开发人员的负担。

### 1.2.2 文件系统阶段

在文件系统阶段，有专门的文件系统对数据进行管理和维护，此时数据可以被长期保存在计算机的磁盘上。应用程序和数据之间由文件系统提供的方法或接口进行调取，这在一定程度上使应用程序和数据之间具备了一定的独立性。此时的程序开发人员可以不必过多地关注数据的存储细节。

但是，文件系统阶段的数据独立性仍然比较低，数据的共享性也比较差，需要进一步提升数据的独立性和共享性。

### 1.2.3 数据库系统阶段

随着互联网技术的发展，计算机产生的数据越来越多，使用文件系统已经远远不能满足各种应用的需求。此时，数据库技术应运而生，标志着数据库的发展进入了数据库系统阶段。

在数据库系统阶段，有专门的数据库管理系统对数据进行管理和维护。此时，应用程序和数据之间由数据库管理系统提供的方法或接口进行调取，所以应用程序和数据之间具备独立性。此时的数据共享性比较好，可以在多个应用程序之间共享，同时，此阶段提供了相应的数据库设计规范，使得数据的冗余度比较低。

在此阶段中，特别是关系型数据库的出现，使得数据能够以一种二维表格的形式进行展现。在关系型数据库中，表里的一行代表着一条完整的基础信息，一列表示数据行的特定属性，大大简化了数据的存储与展现模型。

除了关系型数据库之外，在数据库系统阶段还出现了很多非关系型数据库，如 Key-Value 型数据库 Memcached、Redis 和文档型数据库 MongoDB 等。

### 1.2.4 云数据库阶段

2013 年是大数据元年，标志着互联网正式进入了大数据时代。在大数据时代，互联网会产生更多的数据，传统单机数据库已经无法满足大数据时代对海量数据的存储需求。业界开始探究一种即开即用且具有高度稳定性和可靠性，同时具备可弹性伸缩的在线数据库，以满足大数据时代对海量数据的存储需求。

云数据库可以实现按照需求进行付费使用，按照业务需求对数据库进行扩展；同时，云数据库支持数据的读写分离，数据库发生故障时能够自动切换，自动实现数据的备份操作，实现数据的监控和报警等。目前，已经有越来越多的应用部署到了“云”上，也有越来越多的公司开始使用云数据库。

云数据库可以分为关系型数据库和非关系型数据库两大类。云数据库中的典型代表有阿里巴巴的 RDS 数据库和 OceanBase 数据库。

## 1.3 本 章 总 结

本章简要介绍了数据库的定义，对数据库的相关概念进行了简单描述，然后对数据库的发展进行了简单的介绍。下一章将会对数据库技术进行简要介绍。

# 第 2 章　数据库技术

在某种程度上，数据库不仅是指存储数据的软件系统，它还包括存储数据的硬件。而在存储规模上，数据库的容量可以小到只能存储几 KB 的数据，也能大到存储 GB、TB 甚至是 PB 级别的海量数据。同时，数据库又可以分为关系型数据库和非关系型数据库，而关系型数据库在技术构成上往往又可以分为数据库系统、SQL 语言和数据库访问技术。

本章就对数据库技术进行简单的介绍，主要涉及的知识点如下：

- 数据库系统，了解数据库系统的组成。
- SQL 语言，了解 SQL 语言的定义和组成部分。
- 数据库访问技术，了解数据库中提供的不同的数据访问技术。

## 2.1　数据库系统

在关系型数据库领域中，通常认为数据库系统涉及的软件主要由操作系统、数据库、数据库管理系统、以数据库管理系统为核心的应用开发工具和应用程序等几部分组成。

- 操作系统（Operating System，OS）：直接运行于计算机硬件上的系统，为计算机中运行的各种软件提供基础环境支持。主流的操作系统包括 Windows、UNIX/Linux 和 Mac OS 等。
- 数据库（DataBase，DB）：主要负责数据的存放，并在一定程度上保证数据的安全性、完整性和可靠性。
- 数据库管理系统（DataBase Management System，DBMS）：主要用来对数据库进行管理，是数据库系统的核心组成部分。在实际工作中，人们往往不会直接面对数据库，而是通过数据库管理系统对数据库中的数据进行管理和维护。
- 以数据库管理系统为核心的应用开发工具：为应用开发人员和数据库管理与维护人员提供的高效率、多功能的软件工具集。应用开发人员和数据库管理与维护人员通过以数据库管理系统为核心的应用开发工具，能够更好地开发数据库应用程序，并对数据库进行管理和维护。
- 应用程序：通常由某种或某几种高级编程语言编写，描述用户应用需求的应用程序、软件或某种管理系统。

在某种程度上，除了上述数据库系统涉及的软件之外，数据库管理员（DBA）也可以作

为数据库系统中的一部分。

数据库管理员（DataBase Administrator，DBA）：控制数据库整体结构的人，需要承担创建、管理、监控和维护整个数据库的责任，并保证数据库的安全、完整、高可用性与高可靠性。DBA 可以是一个人，当数据库规模大到一定程度时，DBA 也可以是几个人甚至更多人组成的 DBA 小组。

从 DBA 角度来看，数据库系统整体结构如图 2-1 所示。

从用户角度来看，数据库系统的整体结构如图 2-2 所示。

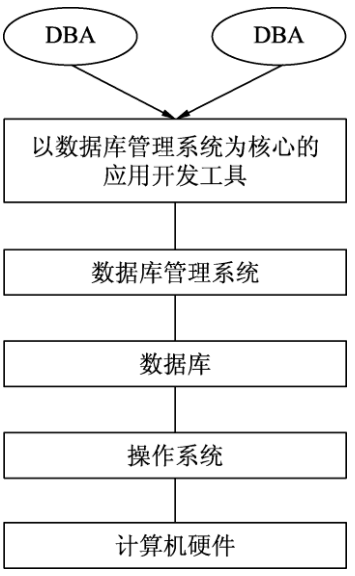


图 2-1　从 DBA 角度看数据库系统

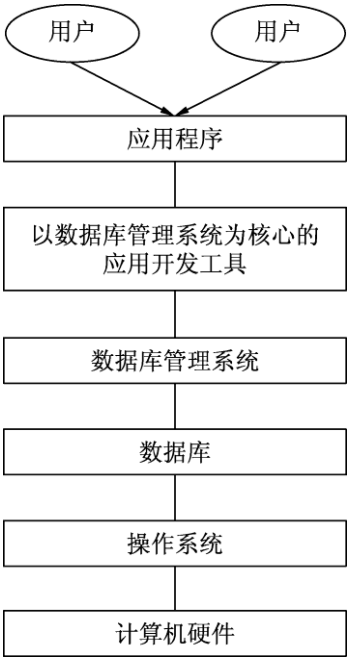


图 2-2　从用户角度看数据库系统

对比图 2-1 和图 2-2 可以得知，从用户角度看数据库系统在结构上比从 DBA 角度看数据库系统多一层“应用程序”。这是因为用户不会直接通过“以数据库管理系统为核心的应用开发工具”与数据库进行交互，而是通过以某种或某几种高级编程语言编写的应用程序、软件或某种管理系统来与数据库进行交互。

例如，用户访问某个网站，网站服务器会记录用户访问的 IP 地址、地理区域、访问接口、用户信息、浏览页面和访问时长等信息，并将这些信息存储到数据库中。当用户访问网站时，用户没有通过“以数据库管理系统为核心的应用开发工具”直接与数据库进行交互，而是由网站服务器对用户的访问信息进行保存。这里的网站就是以某种或某几种高级编程语言编写的应用程序。

通过图 2-1 和图 2-2 还可以看出，整个数据库系统的最底层是由计算机硬件作为基础支撑，运行于计算机硬件之上的是操作系统，而数据库则搭建在操作系统之上。无论是 DBA

还是用户，往往都不会直接操作数据库中的数据，而是通过数据库管理系统提供的界面或者接口来对数据库中的数据进行增、删、改、查等操作。有一定经验的 DBA 为了减轻工作量，同时为了在数据库维护过程中降低出错的成本，往往会依托“以数据库管理系统为核心的应用开发工具”编写数据库维护脚本，来对数据库进行管理和维护。

## 2.2 SQL 语言

关系型数据库中专门提供了一种对数据库进行操作和查询的语言，叫作结构化查询语言，英文为 Structured Query Language，简称 SQL。

### 2.2.1 SQL 语言分类

SQL 语言在功能上主要分为如下 4 类。

- DDL（Data Definition Language，数据定义语言）：用于定义数据库、数据表和列，可以用来创建、删除、修改数据库和数据表的结构，包含 CREATE、DROP 和 ALTER 等语句。
- DML（Data Manipulation Language，数据操作语言）：用于操作数据记录，可以对数据库中数据表的数据记录进行增加、删除和修改等操作，包含 INSERT、DELETE 和 UPDATE 等语句。
- DCL（Data Control Language，数据控制语言）：用于定义数据库的访问权限和安全级别，主要包含 GRANT、REVOKE、COMMIT 和 ROLLBACK 等语句。
- DQL（Data Query Language，数据查询语言）：用于查询数据表中的数据记录，主要包含 SELECT 语句。

下面是一条使用 DDL 语句创建数据表的例子，该语句声明创建一个名为 t_visit_log 的数据表，用来存放用户访问网站的日志信息。

```
CREATE TABLE IF NOT EXISTS `t_visit_log` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `t_user_id` int(11) DEFAULT '0' COMMENT '用户 id',
  `t_ip` varchar(20) DEFAULT '' COMMENT 'IP 地址',
  `t_area` varchar(50) DEFAULT '' COMMENT '区域',
  `t_api` varchar(100) DEFAULT '' COMMENT '访问的接口',
  `t_start_timestamp` bigint(20) DEFAULT '0' COMMENT '开始时间戳',
  `t_end_timestamp` bigint(20) DEFAULT '0' COMMENT '结束时间戳',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户访问日志信息表';
```

该数据表主要包含 7 个字段，各字段的含义分别如下：

- id：t_visit_log 表的主键，INT 类型，设置为自动递增。
- t_user_id：用户 ID，INT 类型。

- t_ip：用户访问网站时浏览器的 IP 地址，VARCHAR 类型。
- t_area：用户所在的区域，VARCHAR 类型。
- t_api：用户访问网站时调用的 API 接口，VARCHAR 类型。
- t_start_timestamp：访问开始时间戳，BIGINT 类型。
- t_end_timestamp：访问结束时间戳，BIGINT 类型。

在 MySQL 数据库命令行中的执行效果如下：

```
mysql> CREATE TABLE IF NOT EXISTS `t_visit_log` (
    -> `id` int(11) NOT NULL AUTO_INCREMENT,
    -> `t_user_id` int(11) DEFAULT '0' COMMENT '用户 id',
    -> `t_ip` varchar(20) DEFAULT '' COMMENT 'IP 地址',
    -> `t_area` varchar(50) DEFAULT '' COMMENT '区域',
    -> `t_api` varchar(100) DEFAULT '' COMMENT '访问的接口',
    -> `t_start_timestamp` bigint(20) DEFAULT '0' COMMENT '开始时间戳',
    -> `t_end_timestamp` bigint(20) DEFAULT '0' COMMENT '结束时间戳',
    -> PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户访问日志信息表';
Query OK, 0 rows affected (0.13 sec)
```

可以看到，使用 DDL 语句创建数据表成功。

接下来使用 DML 语句向 t_visit_log 表中插入数据，如下：

```
mysql> INSERT INTO t_visit_log (id, t_user_id, t_ip, t_area, t_api, t_start_timestamp,
t_end_timestamp) VALUES ('1', '1', '192.168.175.100', '北京市', '/test/api', '1572767893132',
 '1572767894125');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t_visit_log (id, t_user_id, t_ip, t_area, t_api, t_start_timestamp,
t_end_timestamp) VALUES ('2', '2', '192.168.175.101', '广东省深圳市', '/test/api',
'1572767892316', '1572767892643');
Query OK, 1 row affected (0.00 sec)
```

最后使用 DQL 语句查询 t_visit_log 表中的数据，结果如下：

```
mysql> SELECT * FROM t_visit_log;
+---+-----------+-------+---------+----------+-------------------+----------------
+
| id | t_user_id | t_ip  | t_area   | t_api    | t_start_timestamp | t_end_timestamp
|
+---+---+----------------+-----------+-----------+---------------+--------------
+
|  1 | 1 | 192.168.175.100 | 北京市     | /test/api | 1572767893132 | 1572767894125 |
|  2 | 2 | 192.168.175.101 | 广东省深圳市| /test/api | 1572767892316 | 1572767892643 |
+---+---+----------------+-----------+-----------+---------------+--------------
+
2 rows in set (0.00 sec)
```

关于 SQL 语句的简单示例就介绍到这里，在后续的章节中会详细介绍各种 SQL 语句的使用及综合案例。

### 2.2.2 ER 图

关系型数据库提供了 SQL 语言，使应用程序开发人员与数据库管理和维护人员能够与数据库进行交互。但是在创建数据库和数据表之前，需要对数据库中的数据表进行设计，并能够正确设计出各数据表之间的关联关系。

通常使用 ER 图（Entity Relationship Diagram），也就是实体-关系模型，来进行数据表的设计。ER 图是用来描述现实世界的概念模型，在这个模型中有 3 个基本要素，分别为实体、属性和关系。

例如，如图 2-3 所示为部门与员工之间的 ER 图。

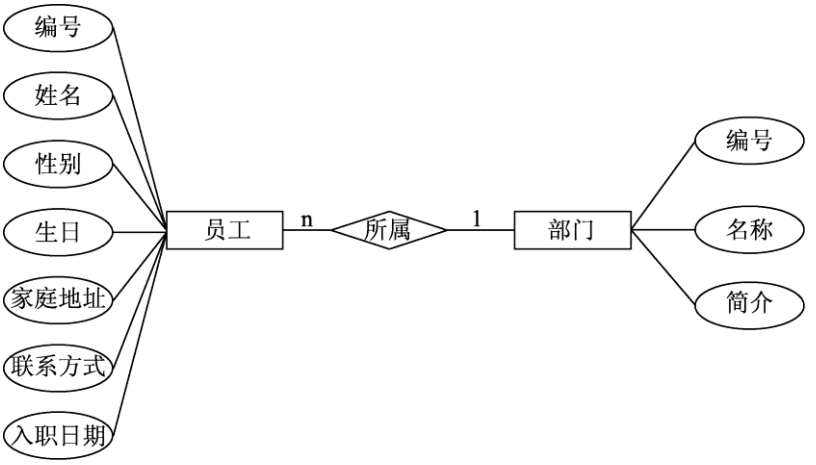


图 2-3　员工与部门之间的 ER 图

如图 2-4 所示为学生选课与教师授课之间的 ER 图。

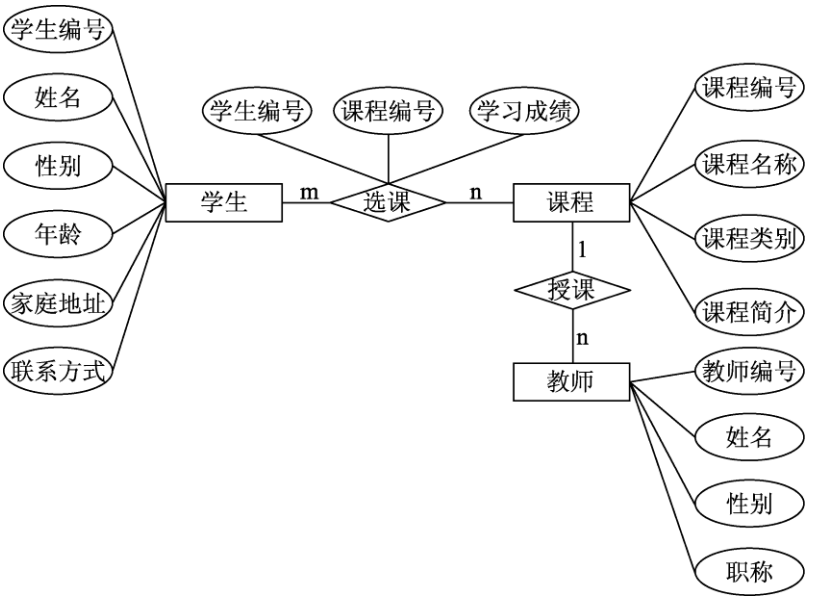


图 2-4　学生选课与教师授课之间的 ER 图

## 2.2.3　SQL 执行流程

当在 MySQL 命令行中输入 SQL 语句并执行时，SQL 语句需要在 MySQL 内部经过一系列的流程，才能将数据操作或查询结果返回给客户端，这一流程可以简化，如图 2-5 所示。

由图 2-5 可知，当向 MySQL 发出 SQL 请求时，一般会经历如下流程。

（1）客户端发送一条 SQL 语句给 MySQL 服务器。

（2）MySQL 服务器先检查查询缓存，如果查询缓存中存在待查询的结果数据，则会立刻返回查询缓存中的结果数据，否则执行下一阶段的处理。

（3）MySQL 服务器通过解析器和预处理器对 SQL 语句进行解析和预处理，并将生成的 SQL 语句解析树传递给查询优化器。

（4）查询优化器将 SQL 解析树进行进一步处理，生成对应的执行计划。

（5）MySQL 服务器根据查询优化器生成的执行计划，通过查询执行引擎调用存储引擎的 API 来执行查询操作。

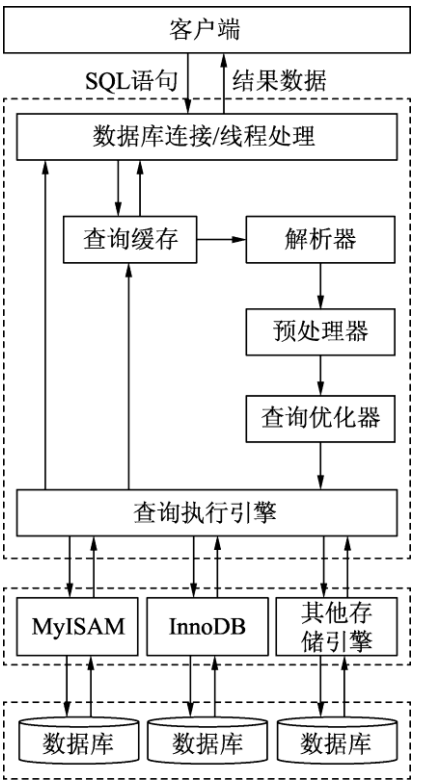


图 2-5　SQL 语句执行流程

（6）存储引擎查询数据库中的数据，并将结果返回给查询执行引擎。

（7）查询执行引擎将结果保存在查询缓存中，并通过数据库连接/线程处理返回给客户端。

这里涉及 MySQL 中的几个名词，说明如下：

- 查询缓存：MySQL 内部的缓存区域，保存查询返回的完整结构，当查询命中缓存时，MySQL 会立刻返回结果数据，不再执行后续的解析、优化和执行操作。
- 解析器：能够通过 SQL 关键字和 SQL 语法规则对 SQL 语句进行解析，并生成对应的 SQL 语句解析树。
- 预处理器：根据 MySQL 的相关规则，对解析器生成的 SQL 语句解析树进行进一步校验，比如检查数据库中的数据表是否存在，数据表中的数据列是否存在。再比如，如果 SQL 语句中使用了别名，还会对别名进行校验，检查别名是否存在重名和歧义等。
- 查询优化器：根据一定的规则将 SQL 语句解析树转化成查询性能最好的执行计划。
- 查询执行引擎：根据查询优化器生成的执行计划完成整个数据查询的过程。

## 2.3 数据库访问技术

不同的编程语言会使用不同的数据库访问技术，对数据库中的数据进行查询和操作。数据库访问技术主要包括 ODBC、DAO、OLE DB、ADO、ADO.NET、JDBC、PDO、PyMySQL、MySQL 2 和 GO-SQL-Driver 等。下面分别对这些数据库访问技术进行简单的介绍。

### 1．ODBC简介

ODBC（Open Database Connectivity，开放数据库互连）是微软公司推出的一种数据库规范，并提供了一组访问数据库的标准 API，由一组函数调用组成，核心是 SQL 语句，但是只针对关系型数据库。

### 2．DAO简介

DAO（Data Access Object，数据访问对象集）是微软公司推出的一种数据库访问技术，它是一种对象集合访问技术，能够独立于数据库管理系统，对数据库进行交互和访问。

### 3．OLE DB简介

OLE DB（Object Linking and Embedding Database，对象连接与嵌入技术）是微软公司推出的一种基于 COM 思想并且面向对象的数据库访问技术。它能够提供统一的数据访问接口，使客户端通过统一的数据访问接口对不同的数据源进行访问，而不必关心数据源的存储位置、存储格式和存储类型等细节信息。

### 4．ADO简介

ADO（ActiveX Data Objects）是微软公司推出的一款用于数据存储的 COM 组件，通过 ADO 连接数据库，它不用关心数据库是如何实现的，能够对数据库中的数据进行灵活的操作和查询。

### 5．ADO.NET简介

ADO.NET 是微软公司推出的全新数据库访问技术，它是从 ADO 数据库访问技术发展而来，并采用了一种全新的技术。值得注意的是，ADO.NET 技术既能在与数据源连接的环境下工作，又能在断开与数据源连接的条件下工作，并封装和隐藏了很多数据库访问的细节信息，使得客户端能够专注于具体的业务处理。

### 6．JDBC简介

JDBC（Java Data Base Connectivity，Java 数据库连接）是为 Java 提供的用于数据库连接和操作的数据访问技术，能够为多种关系型数据库提供统一的访问方式和访问接口，主要由 Java 语言编写的类和接口组成。

### 7．PDO简介

PDO（PHP Data Object）是专门为 PHP 语言设计并推出的数据库访问技术，它屏蔽了底层实现的具体细节信息，对外提供统一的数据访问接口。这样，无论客户端连接的是哪种数据库，均无须关注具体的实现细节，通过调用统一的数据访问接口，即可对数据库中的数据进行操作和查询。

### 8．PyMySQL简介

PyMySQL 是专门为 Python 语言设计并推出的数据库访问技术，主要针对 MySQL 数据库。它能够屏蔽底层复杂的数据库连接与管理，使得 Python 语言能够轻松操作 MySQL 数据库中的数据。

### 9．MySQL 2简介

MySQL 2 是专门为 Ruby 语言设计并推出的数据库访问驱动程序，主要针对 MySQL 数据库，通过 MySQL 2 数据库访问技术，能够使得 Ruby 语言对数据库中的数据进行各种操作。

### 10．GO-SQL-Driver简介

GO-SQL-Driver 是一种专门为 Go 语言设计并推出的数据库访问技术，它能够屏蔽 Go 语言与数据库连接的底层细节信息，使得客户端应用程序不必关心复杂的数据库连接处理过程，而专注于具体的业务操作。同时，Go 语言通过 Go-SQL-Driver 驱动程序提供的统一接口标准，

能够轻松操作数据库中的数据。

## 2.4 本 章 总 结

本章主要对数据库所使用的技术做了简单的介绍，包括数据库系统、SQL 语言和数据库访问技术。然后简单介绍了 SQL 语言的分类、ER 图和 SQL 执行流程。最后对数据库访问技术进行了简单说明，列举了一些不同编程语言经常使用的数据库访问技术和驱动程序。下一章将对 MySQL 数据库进行简单的介绍。

# 第 3 章　MySQL 数据库

MySQL 是一个开源的数据库管理系统。相比于 SQL Server 和 Oracle 数据库管理系统来说，它显得更加小巧和灵活，使用成本也更加低廉。本章将对 MySQL 数据库的三大范式和常用的存储引擎进行简单的介绍。

本章涉及的知识点有：

- MySQL 三大范式；
- MySQL 中的存储引擎。

## 3.1　MySQL 三大范式

MySQL 的三大范式能够规范开发人员对数据表的设计，使得开发人员能够设计出简洁、优雅的数据表结构。

### 3.1.1　第一范式

第一范式主要是确保数据表中每个字段的值必须具有原子性，也就是说数据表中每个字段的值为不可再次拆分的最小数据单元。

例如，表 3-1 所示的 t_user 数据表的设计就不符合第一范式。

表 3-1　不符合第一范式的t_user数据表的设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说　　明 |
|---|---|---|---|
| id | INT | 是 | 主键id |
| username | VARCHAR(30) | 否 | 用户名 |
| password | VARCHAR(50) | 否 | 密码 |
| user_info | VARCHAR(255) | 否 | 用户信息 |

其中，user_info 字段为用户信息，可以进一步拆分成更小粒度的字段，不符合数据库设计对第一范式的要求。将 user_info 拆分后的数据表设计为如表 3-2 所示。

表 3-2　符合第一范式的t_user数据表的设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说　　明 |
|---|---|---|---|
| id | INT | 是 | 主键id |
| username | VARCHAR(30) | 否 | 用户名 |
| password | VARCHAR(50) | 否 | 密码 |
| real_name | VARCHAR(30) | 否 | 真实姓名 |
| phone | VARCHAR(12) | 否 | 联系电话 |
| address | VARCHAR(100) | 否 | 家庭住址 |

表 3-2 所示的数据表设计符合 MySQL 的第一范式。

### 3.1.2　第二范式

第二范式是指在第一范式的基础上，确保数据表中除了主键之外的每个字段都必须依赖主键。例如，表 3-3 所示的数据表设计就不符合第二范式。

表 3-3　不符合第二范式的数据表的设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说　　明 |
|---|---|---|---|
| id | INT | 是 | 商品类别主键id |
| category_name | VARCHAR(30) | 否 | 商品类别名称 |
| goods_name | VARCHAR(30) | 否 | 商品名称 |
| price | DECIMAL(10,2) | 否 | 商品价格 |

由于商品的名称和价格字段不依赖于商品类别的主键 id，所以不符合第二范式。可以将其修改成表 3-4 和表 3-5 所示的表设计。

表 3-4　符合第二范式的t_goods_category数据表的设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说　　明 |
|---|---|---|---|
| id | INT | 是 | 商品类别主键id |
| category_name | VARCHAR(30) | 否 | 商品类别名称 |

表 3-5　符合第二范式的t_goods数据表的设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说　　明 |
|---|---|---|---|
| id | INT | 是 | 商品主键id |
| category_id | VARCHAR(30) | 否 | 商品类别id |
| goods_name | VARCHAR(30) | 否 | 商品名称 |
| price | DECIMAL(10,2) | 否 | 商品价格 |

商品信息表 t_goods 通过商品类别 id 字段 category_id 与商品类别数据表 t_goods_category

进行关联。

### 3.1.3 第三范式

第三范式是在第二范式的基础上，确保数据表中的每一列都和主键字段直接相关，也就是说，要求数据表中的所有非主键字段不能依赖于其他非主键字段。

在第三范式下，需要将表 3-5 所示的 t_goods 数据表进一步拆分成表 3-6 和表 3-7 所示的商品信息表，以及商品信息表与商品类别数据表的关联表。

表 3-6 符合第三范式的t_goods商品信息表

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说 明 |
|---|---|---|---|
| id | INT | 是 | 商品主键id |
| goods_name | VARCHAR(30) | 否 | 商品名称 |
| price | DECIMAL(10,2) | 否 | 商品价格 |

表 3-7 符合第三范式的t_goods_join_category关联表

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说 明 |
|---|---|---|---|
| id | INT | 是 | 关联表id |
| goods_id | INT | 否 | 商品表id |
| category_id | INT | 否 | 商品类别id |

### 3.1.4 反范式化

如果数据库中的数据量比较大，系统的 UV 和 PV 访问频次比较高，则完全按照 MySQL 的三大范式设计数据表，读数据时会产生大量的关联查询，在一定程度上会影响数据库的读性能。此时，可以通过在数据表中增加冗余字段来提高数据库的读性能。

例如，可以将商品信息表设计成表 3-8 所示。

表 3-8 反范式化的t_goods商品信息表设计

| 字 段 名 称 | 字 段 类 型 | 是否是主键 | 说 明 |
|---|---|---|---|
| id | INT | 是 | 商品主键id |
| category_id | VARCHAR(30) | 否 | 商品类别id |
| category_name | VARCHAR(30) | 否 | 商品类别名称 |
| goods_name | VARCHAR(30) | 否 | 商品名称 |
| price | DECIMAL(10,2) | 否 | 商品价格 |

# 3.2 MySQL 存储引擎

存储引擎在 MySQL 底层以组件的形式提供，不同的存储引擎提供的存储机制、索引的存放方式和锁粒度等不同。本节就对 MySQL 中常用的存储引擎进行简单的介绍。

## 3.2.1 查看 MySQL 中的存储引擎

可以在 MySQL 命令行中输入如下命令，查看当前 MySQL 支持的存储引擎。

```
mysql> SHOW ENGINES \G
*************************** 1. row ***************************
     Engine: ARCHIVE
    Support: YES
    Comment: Archive storage engine
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 2. row ***************************
     Engine: BLACKHOLE
    Support: YES
    Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 3. row ***************************
     Engine: MRG_MYISAM
    Support: YES
    Comment: Collection of identical MyISAM tables
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 4. row ***************************
     Engine: FEDERATED
    Support: NO
    Comment: Federated MySQL storage engine
Transactions: NULL
        XA: NULL
  Savepoints: NULL
*************************** 5. row ***************************
     Engine: MyISAM
    Support: YES
    Comment: MyISAM storage engine
```

```
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 6. row ***************************
     Engine: PERFORMANCE_SCHEMA
    Support: YES
    Comment: Performance Schema
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 7. row ***************************
     Engine: InnoDB
    Support: DEFAULT
    Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
        XA: YES
  Savepoints: YES
*************************** 8. row ***************************
     Engine: MEMORY
    Support: YES
    Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
        XA: NO
  Savepoints: NO
*************************** 9. row ***************************
     Engine: CSV
    Support: YES
    Comment: CSV storage engine
Transactions: NO
        XA: NO
  Savepoints: NO
9 rows in set (0.00 sec)
```

结果显示，当前 MySQL 中总共有 9 种存储引擎，除了 FEDERATED 存储引擎外，还支持 8 种存储引擎。

 **注意：**笔者使用的 MySQL 版本为 8.0.18。

### 3.2.2 常用存储引擎介绍

MySQL 中常用的存储引擎有 InnoDB、MyISAM、MEMORY、ARCHIVE 和 CSV，本节对这些存储引擎进行简单的介绍。

### 1．InnoDB存储引擎

InnoDB 存储引擎的特点如下：

- 支持事务。
- 锁级别为行锁，比 MyISAM 存储引擎支持更高的并发。
- 能够通过二进制日志恢复数据。
- 支持外键操作。
- 在索引存储上，索引和数据存储在同一个文件中，默认按照 B+Tree 组织索引的结构。同时，主键索引的叶子节点存储完整的数据记录，非主键索引的叶子节点存储主键的值。
- 在 MySQL 5.6 版本之后，默认使用 InnoDB 存储引擎。
- 在 MySQL 5.6 版本之后，InnoDB 存储引擎支持全文索引。

### 2．MyISAM存储引擎

MyISAM 存储引擎的特点如下：

- 不支持事务。
- 锁级别为表锁，在要求高并发的场景下不太适用。
- 如果数据文件损坏，难以恢复数据。
- 不支持外键。
- 在索引存储上，索引文件与数据文件分离。
- 支持全文索引。

### 3．MEMORY存储引擎

MEMORY 存储引擎的特点如下：

- 不支持 TEXT 和 BLOB 数据类型，只支持固定长度的字符串类型。例如，在 MEMORY 存储引擎中，会将 VARCHAR 类型自动转化成 CHAR 类型。
- 锁级别为表锁，在高并发场景下会成为瓶颈。
- 通常会被作为临时表使用，存储查询数据时产生中间结果。
- 数据存储在内存中，重启服务器后数据会丢失。如果是需要持久化的数据，不适合存储在 MEMORY 存储引擎的数据表中。

### 4．ARCHIVE存储引擎

ARCHIVE 存储引擎的特点如下：

- 支持数据压缩，在存储数据前会对数据进行压缩处理，适合存储归档的数据。
- 只支持数据的插入和查询，插入数据后，不能对数据进行更改和删除，而只能查询。
- 只支持在整数自增类型的字段上添加索引。

### 5．CSV存储引擎

CSV 存储引擎的特点如下：

- 主要存储的是.csv 格式的文本数据，可以直接打开存储的文件进行编辑。
- 可以将 MySQL 中某个数据表中的数据直接导出为.csv 文件，也可以将.csv 文件导入数据表中。

**注意：**笔者只是列举了 MySQL 中常用的一些存储引擎，有关其他存储引擎的知识，读者可以参考 MySQL 官方文档进行了解与学习，网址如下：

https://dev.mysql.com/doc/refman/8.0/en/storage-engines.html

https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html

## 3.3 本 章 总 结

本章主要对 MySQL 中的三大范式和存储引擎的相关知识进行了简单的介绍。下一章将会对如何安装 VMware 虚拟机、Windows 操作系统、Mac OS X 操作系统和 CentOS 操作系统进行简单的介绍。