计算机二级 MySQL 笔记

计算机二级考试官方教材学习笔记

作者:程振兴

网站: https://www.czxa.top

时间: 2019 年 6 月 22 日

版本:第一版

目　　录

第 1 章 导论

这份笔记是我 2018 年夏天花了 15 天(每天学习一章)备考计算机二级 MySQL 的笔记,学习材料就是官方教材。

1.1 本书内容

本书一共有 20 章:

1. MySQL——--心累的入门之旅
2. 数据库技术的基本概念和方法;
3. MySQL 编程语言;
4. 数据定义;
5. 数据查询;
6. 数据更新;
7. 索引;
8. 视图;
9. 触发器;
10. 事件;
11. 存储过程与存储函数;
12. 访问控制与安全管理;
13. 备份和恢复;
14. MySQL 数据库的应用编程;
15. 开发实例;
16. R 连接 MySQL 数据库;
17. Stata 连接 MySQL 数据库的方案;
18. 如何解决 MySQL 导出数据时的权限设置问题?
19. PHP7.2 连接 MySQL8.0 失败的问题解决;
20. 彻底解决了 Stata 与 MySQL 数据交换的问题.

1.2 目标读者

本书的目标读者是闲着没事干的同学。

1.3 许可证

本书是一本开源书籍,使用 Creative Commons Attribution-NonCommercial-NoDerivs 3.0 许可证。这意味着:

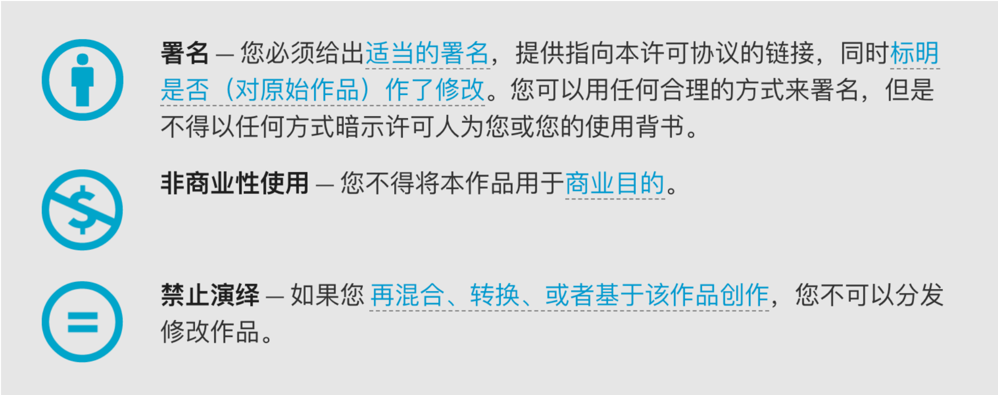


图 **1.1:** Creative Commons Attribution-NonCommercial-NoDerivs 3.0 协议要点

如果你想支持作者的工作，欢迎前往作者的网站对作者进行打赏。你的支持将会促使作者更加及时地更新这本书。

1.4 读者反馈

欢迎读者的反馈。你对本书有任何想法，喜欢或者不喜欢什么，请告知我。你可以在下面的评论区里评论，如果你阅读的是 PDF 版本，你可以前往 czxa/mysql-notes 创建 issues。

第 2 章 MySQL——--心累的入门之旅

今天中午取到了之前买的 MySQL 计算机二级考试的教材书，下午考完试就开始了 MySQL（Mac 版本）的入门之旅，非常心累。

下面记录了这次探索的过程。

2.1 安装 MySQL 社区版

点击下面的下载地址即可跳转下载。下载地址

2.2 安装一个 IDE

虽然我的电脑上装了一大堆我不会有的数据库管理的软件，但是出于对 PyCharm 和 CLion 的喜爱，我还是选择了 DataGrip，破解版可以到这里下载：DataGrip。

2.3 登录 MySQL

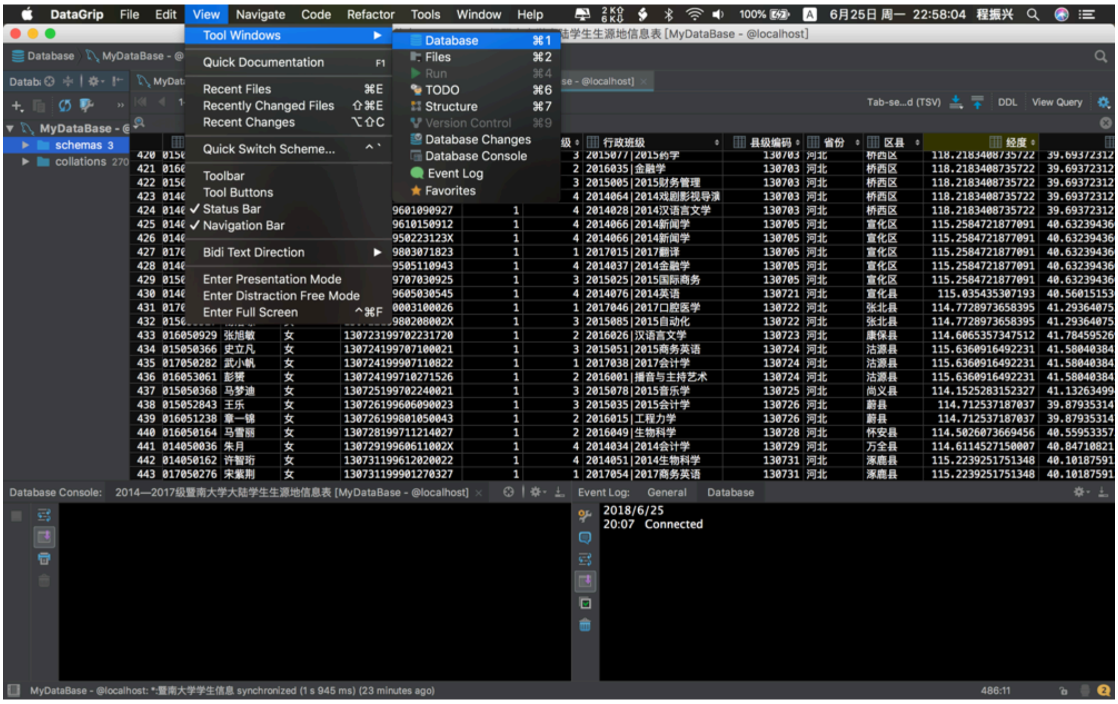
参考这篇推文解决 Access denied for user 'root'@'localhost' (using password: NO)。登录方法如下：

```
# 登录
mysql -u root -p;
# 输入默认密码后，会登陆成功；
# 执行下面的命令可以设置新的密码：
SET PASSWORD = PASSWORD("XXXXX");
# 退出
exit;
```

2.4 DataGrip 连接 MySQL

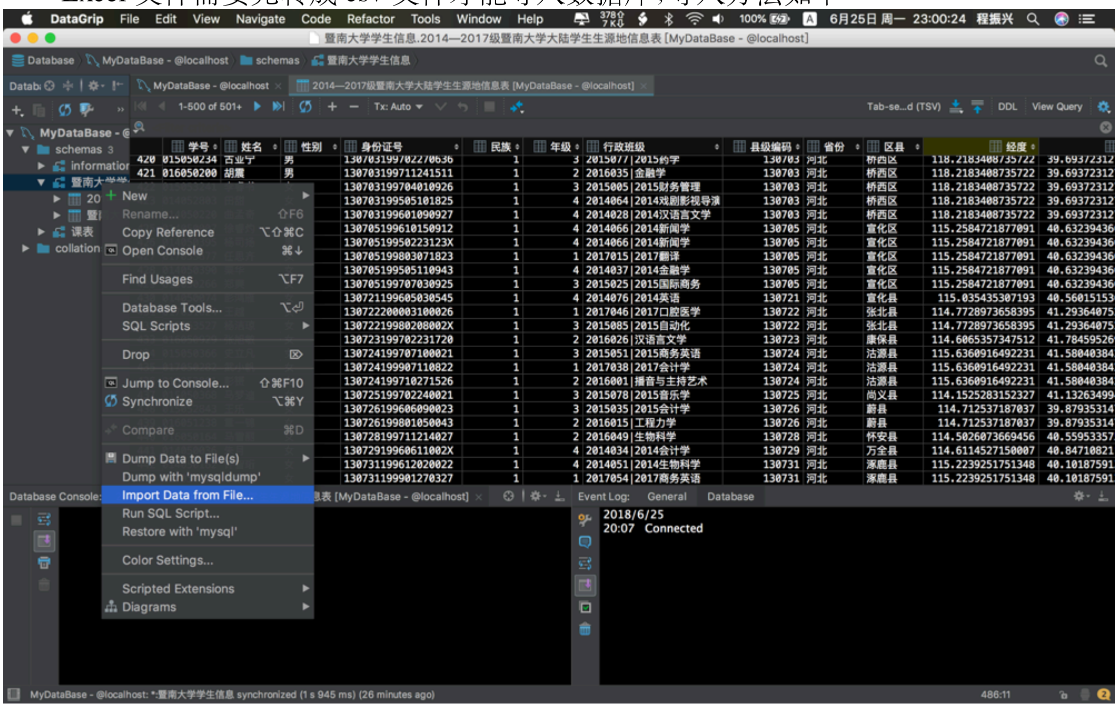
刚刚打开 DataGrip 的时候我是非常懵逼的，最后研究了好一会儿才基本搞明白，具体可以看考这篇：Data Grip 使用—> 创建数据库连接

另外，点击这个就会出现这个数据库的结构：



2.5 将 Excel 文件导入数据库

Excel 文件需要先转成 csv 文件才能导入数据库，导入方法如下：



大概就是这些了。

2.6 Python 连接数据库

最后又研究了一下 Python 如何链接数据库。需要 pymysql 包，下面的两行代码即可连接数据库，不过会出错：

```
import pymysql
db = pymysql.connect("localhost","root","你的密码","MyDataBase" )
```

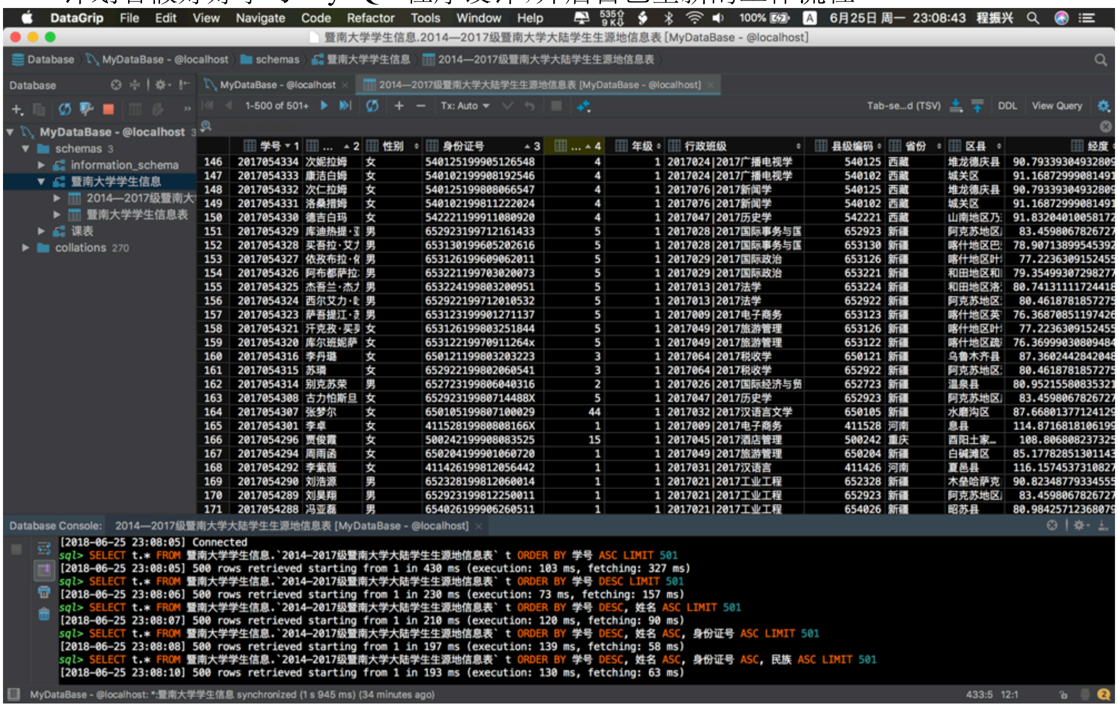
出错的解决方法可以参考下面的这个文章：pymysql pymysql.err.OperationalError 1045 Access denied 最简单解决办法。

2.7 探索失败的地方

本来是想再安装一下 ODBC 的，这样就可以跟 Stata 一起使用了。但是 Mac 版本的不像 Windows 的那样好用，尝试了很久也没成功，最后也不得不放弃。

2.8 未来计划

计划暑假好好学习 MySQL 程序设计，开启自己全新的工作流程！



第 3 章 数据库技术的基本概念和方法

下个月就要考二级 MySQL 了,终于要开始学习了!

3.1 数据库基础知识

3.1.1 数据库相关的基本概念

1. 数据:描述事物的符号记录。数据的解释是对数据含义的说明,也被称为数据的语义。数据与其语义密不可分,没有语义的数据是没有意义和不完整的。
2. 数据库:是长期存储在计算机内、有组织、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和存储,具有较小的冗余度、较高的数据独立性,系统易于扩展,并可被多个用户共享。
3. 数据库管理系统 (DataBas Management System, DBMS):是位于操作系统和用户之间的一层数据管理软件,是数据库系统的核心。DBMS 是按照一定的数据模型科学地组织和存储数据,能够高效地获取数据,提供安全性和完整性等统一控制机制,有效地管理、维护数据,是数据库系统的核心。DBMS 的主要功能包括数据定义、数据操纵、数据库的建立、数据库的运行和管理等。
   - 数据定义:DBMS 提供数据定义语言 (Data Definiton Language, DDL)。用户通过 DDL 可以对数据库中的数据对象进行定义。
   - 数据操纵:DBMS 提供数据操纵语言 (Data Manipulation Language, DML)。使用 DML,能够操纵数据,实现对数据库的基本操作,例如数据的查询、插入、删除和修改等。
   - 数据库的建立和维护功能:主要包括数据库初始数据的输入、转换、数据库的转储、恢复、数据库的重组织和性能监视、分析等。这些功能通常是由一些实用程序来完成的。
   - 数据库的运行管理:数据库的建立、运用和维护等工作由 DBMS 统一管理、统一控制,以保证数据的安全性、完整性、多用户对数据的并发使用及发生故障后的系统恢复。
   - 提供方便、有效存取数据库信息的接口和工具:编程人员可以通过程序开发工具与数据库接口编写数据库应用程序。数据库管理员 (DataBase Administrator, DBA) 可以通过相应的软件应用工具对数据库进行管理。
4. 数据库系统 (DataBase System, DBS) 是指引入数据库技术的计算机系统。一个完整的数据库系统不仅仅包括数据库,还包括支持数据库的硬件、数据库管理系统及相关软件、数据库管理员和用户。在不引起混淆的情况下,常常将数据库系统简称为数据库。

3.1.2 数据库系统的特点

1. 数据结构化。数据库系统中的数据不再针对某一应用，而是面向全局应用，具有整体的结构化。这里所说的结构是指数据的组织方式。不仅数据是结构化的，而且存取数据的方式也很灵活，可以存取数据库中的某一个数据项、一组数据项、一个记录或一组记录，并且数据库中的结构化数据由 DBMS 统一管理。DBMS 既管理数据的物理结构，还管理数据的逻辑结构；既考虑数据的本身，也考虑数据之间的联系。
2. 数据冗余度小。数据库系统从整体和全局上看待和描述数据，数据不仅面向某个应用，而且面向全局应用，从而可大大减少数据冗余，节约存储空间，避免因数据的重复存储和不同拷贝而造成数据之间的不一致性。
3. 数据共享性好。同样，由于数据库系统是从整体和局部上看待和描述数据，使得数据不仅仅面向某个应用，而且面向整个系统，因此数据可以被多个用户和多个应用共享使用。数据共享可以大大减少数据冗余，节约存储空间，还可以避免由于数据冗余造成的同一数据重复存储而导致修改时的困难和可能造成数据的不一致。
4. 数据的独立性高。数据的独立性包括数据的物理独立性和数据的逻辑独立性。物理独立性是指用户的应用程序与存储在磁盘上数据库中的数据是相互独立的，逻辑独立性则是指用户的应用程序与数据库的逻辑结构是相互独立的。也就是说，数据的逻辑结构改变了，用户的程序也可以不变。数据的独立性是数据库的一种特征和优点，它有利于在数据库结构发生改变时保持应用程序尽可能地不改变或减少改变，这样可以减少应用人员的开发工作量。
5. 数据库的保护。数据库管理系统具有对数据的统一管理和控制功能，主要包括数据的安全性、完整性、并发控制和故障恢复等，既数据库的保护。
   - 数据的安全性是指保护数据，以防止不合法的使用而造成数据的泄密和破坏，使每个用户只能按照规定对某些数据以某些方式进行使用和处理，即保证只有赋予权限的用户才能访问数据库中的数据，防止对数据的非法使用。
   - 数据的完整性。是指对数据的正确性、有效性和相容性的要求，即控制数据在一定的范围内有效或者要求数据之间满足一定的关系，保证输入到数据库中的数据满足相应的约束关系，以确保数据有效、正确。
   - 并发控制。是指对多个用户的并发进程同时存取、修改数据库时，可能会发生相互干扰而得到错误的结果，并使得数据库的完整性遭到破坏，因而对多用户的并发操作加以控制和协调。
   - 故障恢复。计算机产生的硬件故障、操作员的失误以及人为的破坏都会影响数据库中数据的正确性，甚至造成数据库部分或全部数据的丢失，DBMS 必须具有将数据库从错误中恢复到某一已知的正确状态的功能，这就是数据库的故障恢复。

3.1.3 数据库系统的结构

从数据库用户视图的视角来看，数据库系统通常采用三级模式结构，这就是数据库管理系统 ；若从数据库管理系统的角度来看，数据库系统的结构分为集中式结构、分布式结构、客户/服务器结构和并行结构，这就是数据库系统的 ；若从数据库系统应用的角度来看，目前的数据库系统常见的结构有 / 和 / ，这是数据库系统 。

1. 客户/服务器 (C/S) 模式。客户端和服务器可以同时工作在同一台计算机上，这种工作方式被称为 ‘‘单机方式’’；也可以 ‘‘网络方式’’ 运行，即服务器被安装和部署在网络中的某一台机器上，而客户端被安装和部署在网络中不同的一台或多台主机上。客户端应用程序的开发，目前常用的工具主要有 VC++、NET 框架、Delphi、VB 等。
2. 浏览器/服务器 (B/S) 模式。基于 Web 应用的客户服务器结构，也被称为三层客户/服务器结构。目前，开发基于 B/S 结构的数据库应用系统，主要使用的开发语言有 PHP、Java、Peal、C# 等。

3.1.4 数据模型

根据不同的应用目的，数据模型可以分为概念模型、逻辑模型和物理模型。1. 概念模型（Conceptual Model）用来描述现实世界的事物，与具体的计算机系统无关。现实世界是存在与人脑之外的客观世界。在设计数据库时。用概念模型来抽象表示现实世界的各种事物及其联系。最典型的概念模型是实体联系 (Entity-Relationship, E-R) 模型。E-R 模型用 E-R 图表示。2. 逻辑模型（Logical Model）是具体的 DBMS 所支持的数据模型。任何 DBMS 都基于某种逻辑模型。主要的逻辑模型有层次模型（Hierarchial Model）、网状模型（Network Model）、关系模型（Relational Model）、面向对象模型（Object-Oriented Model）等。3. 物理模型是用于描述数据在存储介质上的组织结构。每一种逻辑数据模型在实现时都有与其相对应的物理数据模型。物理数据模型不但由 DBMS 的设计决定，而且与操作系统、计算机硬件密切相关。物理数据结构一般都向用户屏蔽，用户不必了解其细节。

3.2 关系数据库

关系数据库是目前应用最为广泛的数据库，它以关系模型作为逻辑数据模型，采用关系作为数据的组织方式，其数据库操作建立在关系代数的基础上，具有坚实的数据基础。关系数据库具有较高的数据独立性，当数据的存储结构发生变化时，不会影响应用程序，这样，能大大减少系统维护的工作量。

3.2.1 基本概念

关系的数据结构就是二维表。无论是实体还是实体之间的联系都用关系表示。从用户的角度来看，关系数据库以二维表格的形式组织数据。

1. 表：也被称为关系。实际的数据库应用系统中，一般不采用中文作为表名、字段名等。

2. 列(Field):也称为字段或属性/表中的每一列都有一个名称,称为字段名、属性名或列名。
3. 行(Row):也被称为元组或记录。
4. 关键字(key):是表中能够唯一确定一个元组的属性或属性组。关键字也被称为码或主键。
5. 候选键:如果一个表具有多个能够唯一标识一个元组的属性,则这些属性被称为候选键。
6. 外部关键字:也被称为外键。如果表的一个字段不是本表的主键或候选键,而是另外一个表的主键或候选键,则该字段称为外键。
7. 域:表示属性的取值范围。
8. 数据类型:限制或容许该列中存储的数据。

3.2.2 基本性质

1. 关系必须满足最基本的要求,即:每一列都必须是不可再分的数据项;
2. 表的任意两个元组不能完全相同。即使完全相同的记录,在数据库中必须予以区别;
3. 表中的每一列是同一数据类型,且列的值来自相同的域;
4. 不同列的值可以出自同一个域,但是列名不能完全相同;
5. 表中列的顺序可以任意交换,行的顺序也可以任意交换。

第 4 章 MySQL 编程语言

MySQL 服务器正确安装完毕之后，就在机器系统上搭建好了一个完整的 DBMS，此时数据库的使用者就可以通过命令行或者 GUI 界面等多种客户端实用工具，来建立与 MySQL 数据库服务器的连接，从而实施各种数据库相关的操作。然而，不论使用哪种方式的客户端工具，其与 MySQL 数据库服务器交互的实质，都是通过结构化查询语言（Structured Query Language, SQL）来实现的，所以 SQL 是各类数据库交互方式的基础。因而本书主要采用命令行纯文本的方式重点介绍 MySQL 数据库交互过程中常用的 SQL 语句的语法及使用。

4.1 结构化查询语言 SQL

与其他程序语言不同，SQL 由很少的词构成，这些词被称为关键字，每一个 SQL 语句都由一个或多个关键字关键字组成。设计 SQL 的目的就是要能够很好的提供一种从数据库读写数据的简单而有效的方法。SQL 具有如下优点：

1. SQL 不是某个特定数据库供应商专有的语言，所有的关系型数据库都支持 SQL；
2. SQL 简单易学。它的全部语句都是由描述性很强的英语单词组成，而这些英语单词的数目并不多。
3. SQL 尽管看上去非常简单，但是它实际上是一种强有力的语言，灵活使用其他语言元素可以进行非常复杂和高级的数据库操作。

当前，尽管许多 DBMS 供应商都通过增加语句或指令的方式，对 SQL 进行了各自的扩展，但它们都遵循标准 SQL，即 ANSI SQL，并以标准 SQL 为主体，MySQL 也不例外。它们的这种扩展的主要目的是提供执行特定操作的额外功能或简化方法。因此，读者对于 MySQL 数据库交互操作的学习，应以 SQL 为基础。

另外需要指出的是，SQL 语句不区分大小写。许多 SQL 开发人员习惯对于所有的 SQL 关键字使用大写，而对所有的列和表的名称使用小写，这样的书写方式可以使代码更易于阅读和调试。（因为我是中国人，使用大写非常麻烦，大写切换需要先切换到英语再切换到英语大写，另外自己对大写的英语单词不敏感，所以我决定先使用小写，日后熟练了再改用大写。）

4.2 MySQL 语言的组成

MySQL 数据库所支持的 SQL 语言主要包含以下几个部分：1. 数据定义语言（DDL）：数据定义语言主要用于对数据库及数据库中的各种对象进行创建、删除、修改等操作。其中，数据库对象主要有表、默认约束、规则、视图、触发器、存储过程等。数据定义语言包括的主要 SQL 语句有以下三个：- create：用于创建数据库或数据库对象；- alter：用于对数

据库或数据库对象进行修改;- drop:用于删除数据库或数据库对象。对于不同的数据库对象,这三个 SQL 语句分别说使用的语言格式是不同的。

1. 数据库操纵语言(DML):数据库操纵语言主要用于操纵数据库中的各种对象,特别是检索和修改数据。数据库操纵语言包括的主要 SQL 语句如下:
   - select:用于从表或视图中检索数据,是使用最为频繁的 SQL 语句之一;
   - insert:用于将数据插入到表和或视图中;
   - update:用于修改表或视图中的数据,其即可修改表或视图中的一行数据,也可以修改多行或全部数据;
   - delete:用于从表或视图中删除数据,其中可以根据条件删除指定的数据。
2. 数据控制语言(DCL):数据控制语言主要用于安全管理, 例如确定哪些用户可以查看或修改数据库中的数据。数据控制语言包括的主要 SQL 语句如下;
   - grant:用于授予权限,可把语句许可或对象许可的权限授予其他用户和角色;
   - revoke: 用于收回权限, 其功能与 grant 相反, 但不影响该用户或角色从其他角色中作为成员继承的许可权限。
3. MySQL 扩展增加的语言要素这部分不是标准的 SQL 所包含的内容,而是为了用户编程的方便所增加的语言要素。这些语句包括常量、变量、运算符、表达式、函数、流程控制语句等。
   - 常量:
     - 字符串常量: 是指用单引号或双引号括起来的字符序列, 分为 ASCII 字符串变量和 Unicode 字符串变量。
     - 数值常量:可分为整数常量和浮点数常量。
     - 十六进制常量: 一个十六进制值通常指定为一个字符串常量, 每对十六进制数字被转换为一个字符, 其最前面有一个大写字母 "X'' 或小写字母 "x''。
     - 日期时间常量:是用单引号将表示日期时间的字符串括起来而构成的。
     - 位字段值: 可以用 `b'value'` 格式的符号书写字段值。其中, value 是用 0 或 1 书写的二进制值。位字段符号可以方便地指定分配给 BIT 列的值。
     - 布尔值:只包含两个可能的值,TRUE 和 FALSE。其中 FALSE 的数字值为 0,TRUE 的数字值为 1。
     - NULL 值: 通常用于表示 ''没有值"''无数据'' 等含义, 它与数值类型的 0 或字符串类型的空字符是完全不同的。
   - 变量:在 MySQL 中,变量分为用户变量和系统变量。在使用时,用户变量前常添加一个符号 @,用于将其与列名区分开;而大多数系统变量应用于其他 SQL 语句中,@。
   - 运算符:
     - 算术运算符:+ - * / %;
     - 位运算符:&`` | ``^( ) ~`` >>( ) <<( );
     - 比较运算符:= > < >= <= <>( ) != <=>( )

    - 逻辑运算符:NOT !! AND && OR || XOR(   )
- 表达式:
    - 表达式是常量、变量、列名、复杂计算、运算符和函数的组合。一个表达式通常可以得到一个值。与常量、变量一样,表达式的值也具有某种数据类型。
- 内置函数:
- MySQL 中包含了 100 多个函数,大致可以分为这样几类:
    - 数学函数:abs()、sort();
    - 聚合函数:count();
    - 字符串函数:ascii()、char();
    - 日期和时间函数:now()、year();
    - 加密函数:encode()、encrypt();
    - 控制流程函数:if()、ifnull();
    - 格式化函数:format();
    - 类型转换函数:cast();
    - 系统信息函数:user()、version();

4.3 MySQL 函数

4.3.1 聚合函数

4.3.1.1 count()

count() 函数,对于除 * 以外的任何参数,返回所选择集合中非 null 值的行的数目;对于参数 *,则返回所选择集合中所有行的数目,包含 null 值的行。没有 where 子句的 count(*) 是经过内部优化的,能够快速地返回表中所有的记录总数。

```
# count()函数
use 交易数据;
select count(*) from 000001大单交易数据;
```

上面的语句会返回这个表格的行数(记录数)。

4.3.1.2 sum()

sum() 函数可以用来统计某一字段的列和。

```
# sum()函数
select sum(volume) from `000001大单交易数据`;
```

4.3.1.3 avg()

平均值。

```
# avg()函数
select avg(volume) from `000001大单交易数据`;
```

4.3.1.4 max()

最大值。

```
# max()
select max(volume) from `000001大单交易数据`;
```

4.3.2 数学函数

```
# 绝对值函数: abs()
select abs(5), abs(-5)
-- 5, 5
# 下整函数: floor()
select floor(1.5), floor(-1.5);
-- 1,  2
# 上整函数: ceil()
select  ceil(1.5), ceil(-1.5);
-- 2, -1
# 0~1之间的随机数: rand()
select rand();
-- 0.7719657540060261
# 保留指定位数的小数: truncate(x, y)
select truncate(3.1415926, 2);
-- 3.14
# 平方根函数: sqrt()
select sqrt(5);
-- 2.23606797749979
select truncate(sqrt(5), 2);
-- 2.23
```

4.3.3 字符串函数

```
# 大写函数: upper()/ucase()
select upper('hello'), ucase('hello');
-- HELLO, HELLO
# 子字符串函数(取前n个): left(s, n)
select left('hello', 2);
-- 'he'
# 子字符串函数(从第n个位置取len个): substring(s, n, len)
select substring('hello', 2, 2)
-- 'el'
```

4.3.4 日期函数

```
# 获取当前日期: curdate()/current_date()
select curdate(), current_date;
-- 2018-08-18, 2018-08-18
# 获取当前时间: curtime()/current_time()
select curtime(), current_time();
-- 15:15:08, 15:15:08
# 获取当前日期和时间: now()
select now();
-- 2018-08-18 15:16:15
# 获取当前日期和时间的一些函数
select now(), current_timestamp(), localtime(), sysdate(), localtimestamp();
--2018-08-18 15:18:30    2018-08-18 15:18:30 2018-08-18 15:18:30 2018-08-18
    15:18:30 2018-08-18 15:18:30
```

4.3.5 其它函数

4.3.5.1 if(expr, v1, v2)

表达式成立执行 v1,反之执行 v2。

```
# 查询表`全部个股当日数据`, 如果trade字段的值大于100, 这输出字段level为'优
    秀', 反之为'一 般':
use 交易数据;
select code, name, if(trade > 100, '优秀', '一般') level from 全部个股当日数
    据;
-- code    name   level
-- 603999    读者传媒    一般
-- 603998    方盛制药    一般
-- 603997    继峰股份    一般
-- ······
```

4.3.6 ifnull(v1, v2)

表示如果表达式 v1 不为空则显示 v1 的值,否则显示 v2 的值。

```
select ifnull(1/0, '空');
-- '空'
```

4.3.7 version() 函数

version 函数是一种系统信息函数,用于获取数据库的版本号:

```
select version();
-- 8.0.11
```

第 5 章 数据定义

数据库可看作一个专门存储数据对象的容器，这里的数据对象包括表、视图、触发器、存储器过程等，其中表是最基本的数据对象。

5.1 定义数据库

5.1.1 创建数据库

创建数据库的基本语法格式是：

```
create {datebase|schema} {if not exists} db_name
  [[default] character set [=] charset_name]
  [ [default] collate [=] collaction_name]
```

- [ ] 内为可选项；
- | 用于分隔花括号中的选择项，表示可任选其中一项来与花括号外的语法成分共同组成 SQL 语句命令；
- db_name：数据库名，在文件系统中，MySQL 的数据库存储区将以目录的方式表示 MySQL 数据库。因此命令中的数据库名字必须符合操纵系统中文件夹的命令规则，而在 MySQL 中是不区分大小写的。
- if not exists:判断不存在的时候才会新建。
- character set:用于指定数据库字符集，charset_name 为字符集的名字。简体中文的字符集名称为 gb2312。
- collate:用于指定字符集的校对规则，collaction_name 为校对规则的名称。简体中文字符集的校对规则为 gb2312_chinese_ci。
- default:指定默认的数据库字符集和字符集的校对规则。

例如，创建一个名为 db_school 的数据库，输入语句如下：

```
create database db_school
  default character set GB2312
  default collate gb2312_chinese_ci;
```

上面的命令再次运行的时候是会报错的。MySQL 不允许统一系统中存在相同名称的数据库，在上面的命令中加入 if not exists 可以避免报错。

```
create database if not exists db_school
  default character set GB2312
  default collate gb2312_chinese_ci;
```

用户在使用 create database 或 create schema 创建数据库时必须获得相应的权限。

5.1.2 选择和查看数据库

5.1.2.1 选择数据库

使用 use 命令可以从一个数据库跳转到另一个数据库。在创建数据库之后,该数据库不会自动成为当前数据库,可使用 use 命令指定其为当前数据库。

```
use db_school;
```

5.1.2.2 查看数据库

下面两个语句的作用是一样的,可以用来查看当前用户权限内所能看到的数据库名称。

```
# 查看数据库
show databases;
show schemas;
```

系统数据库有下面四个:

1. mysql:描述用户访问权限;
2. information_schema:保存关于 MySQL 服务器所维护的所有其它数据库信息,如数据库名、数据库的表、表字段、表字段的数据类型与访问权限等;
3. performance_schema:主要用于收集数据库服务性能参数;
4. test:用户利用该数据库进行测试工作。

5.1.3 修改数据库

在 MySQL 中,数据库的默认字符集为 `latin1`,默认校对规则为 `latin1_swedish_ci`。使用 `alter database` 或 `alter schema` 语句可以修改。其语法格式为:

```
alter {database | schema} [db_name]
  [default] character set [=] charset_name
  [default] collate [=] collation_name;
```

- 该命令可以用于修改数据库的全局特性,这些特性储存在数据库目录中的 `db.opt` 中;
- 使用该命令时,用户必须具有对数据库进行修改的权限;
- 数据库的名称可以被省略,表示修改当前数据库。

```
alter database db_school
  default character set gb2312
  default collate gb2312_chinese_ci;
```

5.1.4 删除数据库

基本语法格式为:

```
drop {database | schema} [if exists] db_name;
```

- 这个删除是永久性的;
- 该命令需要相应的权限;
- 但某个数据库被删除后,该数据库的用户权限不会自动删除,为了方便数据的维护,应手动删除它们;
- 可以用 `if exists` 避免删除不存在的数据库是出现 MySQL 错误信息。

```
drop database db_school;
drop database if exists db_school;
```

5.2 定义表

5.2.1 MySQL 中常用的数据类型

MySQL 的常用数据类型主要有数值型、日期和时间型、字符串型。

5.2.1.1 数值类型

- bit[(M)]:位字段类型。M 表示每个值的位数,范围位 1~64。如果 M 被省略,默认为 1。
- tinyint[(M)] [unsigned] [zerofill]: 很小的整数。带符号的范围为-128~127, 不带符号的范围为 0~255。
- bool, boolean:是 tinyint(1) 的同义词。zero 值被视为假,非 zero 值被视为真。
- smallint[(M)] [unsigned] [zerofill]:小的整数。
- mediumint[(M)] [unsigned] [zerofill]:中等大小的整数
- int[(M)] [unsigned] [zerofill]:普通整数。
- integer[(M)] [unsigned] [zerofill]:int 的同义词。
- bigint[(M)] [unsigned] [zerofill]:大整数。
- double[(M)] [unsigned] [zerofill]:双精度浮点数,M 是小数总位数,D 是小数的后面的位数。如果 M 和 D 被省略,根据硬件运行的限制来保存值。双精度浮点数精确到大约 15 位小数。如果指定 unsigned 则不允许负值。
- decimal[(M[, D])] [unsigned] [zerofill]:压缩的 ‘‘严格定点数’’。M 位小数位数(精度)的总数,不包括负号和小数点。D 是小数点(标度)后面的位数。decimal 整数的最大位数 M 位 65,支持的十进制的最大位数 D 是 30。如果 D 被省略,默认为 0。如果 M 被省略,默认为 10。如果指定 unsigned 则不允许负值。
- dec:decimal 的同义词。

5.2.1.2 日期类型

- date: 日期型, MySQL 中以 “YYYY-MM-DD’’ 格式显示 date 的值。但是允许使用字符串或数字为 date 列分配值。
- datetime: 日期和时间的组合, MySQL 中以 “YYYY-MM-DD HH:MM:SS’’ 的格式显示 datetime 的值, 但是允许使用字符串或数字为 datetime 列分配值。
- timestamp[(M)]: 时间戳。范围是 ‘1970-01-01 00:00:00’ 到 2037 年。timestamp 列用于 insert 或 update 操纵时记录日期和时间。如果不分配一个值，表中的第一个 timestamp 列自动设置为当前的日期和时间。timestamp 值返回后显示为 “YYYY-MM-DD HH:MM:SS’’ 格式的字符串, 显示宽度固定为 19 个字符。如果想要获得数字值, 应在 timestamp 列添加 +0。
- time:时间型。
- year[(2|4)]: 两位或者四位格式的年。默认为四位格式。在四位格式中，允许的值为 1901~2155 和 0000 。在两位格式中, 允许的值为 70~69, 表示 1970~2069。

5.2.1.3 字符串类型

- char[(M)]:固定长度的字符数据类型, 用于保存以文本格式存储的信息。但保存时在右侧填充空格以达到指定的长度。M 表示列长度, 其范围为 0~255。
- varchar[(M)]: 可变长度的字符串, 用以保存以文本格式存储的信息。M 表示最大列长度, 其范围为 0~65535。
- tinytext:最大长度为 255 字符的 text 列。
- text:最大长度为 65535($2^{16}-1$)字符的 text 列。

5.2.2 创建表

在 MySQL 中, 创建数据表使用 create table 语句, 其基本的语法格式如下:

```
create table tbl_name
  (
    字段1 数据类型 [列级完整性约束条件] [默认值],
    字段2 数据类型 [列级完整性约束条件] [默认值]
    [, ······]
    [, 表级完整性约束条件]
  )[engine = 引擎类型];
```

例如, 在已有的数据库 db_school 中构建学生表 tbl_student, 要求使用 InnoDB 引擎存储表数据。其结构如表 5.1:

```
use db_school;
create table tb_student
(
  studentNo char(10) not null unique,
  studentName char(20) not null,
  sex char(2),
```

表 5.1: InnoDB 引擎存储表数据结构

含义	字段名	数据类型	宽度
学号	student_No	字符型	10
姓名	student_Name	字符型	20
性别	sex	字符型	2
出生日期	birthday	日期型	
籍贯	native	字符型	20
民族	nation	字符型	30
所属班级	classNo	字符型	6

```
  birthday date,
  native varchar(20),
  nation varchar(30),
  classNo char(6)
) engine = InnoDB;
```

使用 create table 语句创建表时的主要语法说明如下：

1. 表名是必须的，必须符合命名规则。表的创建需要首先选定当前数据库，如果表名被指定为 dbname.tblname 的格式，则可以在特定的数据库创建表，而不论是否为当前数据库，都可以以这种方式创建数据表。如果使用加引号的识别名方式，则应该在数据库的名称和表的名称上都加上引号，例如：'dbname'.'tblname' 是合法的，但是'dbname.tblname' 是不合法的。
2. 完整性约束条件。包括实体完整性约束 (primary key、unique)、参照完整性约束 (foreign key) 和用户自定义约束 (not null、default、check 约束等)。但用户操作表中的数据是，DBMS 会自动检查该操作是否遵循这些完整性约束。
3. null 和 not null。允许 null 的列也允许插入记录时不给出该列的值。not null 值的列则不接受该列没有值的记录。
4. auto_increment。将字段设置为自增属性可以给记录一个唯一而又容易确定的 ID 号，该字段可以唯一标识表中的每条记录。只有整型列才能设置这个属性。当往一个被定义为 auto_increment 的列中插入 null 或者 0 时，该列的值会被 value 加 1 递增。其中 value 为当前表中最大的值。每个表只能定义一个 auto_increment 列，并且必须在该列上定义主键约束 (primary key) 或者候选键约束 (unique)。

例如，如果在上面的例子中，定义学号自动递增，则学号的数据类型必须为整型，而且必须在学号上定义 primary key 或 unique：

```
use db_school;
drop table if exists tbl_student;
create table if not exists tbl_student
(
  studentNo int(10) not null unique auto_increment,
  studentName varchar(20) not null,
  sex char(2),
```

```
  birthday date,
  native varchar(20),
  nation varchar(30),
  classNo char(6)
)engine = InnoDB;
```

当一个列被指定为 auto_increment 后，其值是可以被覆盖的，即可以简单地在数据插入语句（insert 语句）中为该列指定一个值后。只要该值是唯一的，那么这个值将被用来替代系统中自动生成的值，并且后续的增量将基于该手工插入的值。

1. default。默认值是指在向数据表中插入数据时，如果没有明确给出某个字段所对应的值，则 DBMS 此时允许为此字段指定一个值。例如，如果汉族同学比较多，那么在学生表 tbl_student 中可将该字段的默认值设置为 ‘‘汉’’。当插入一条新记录时，如果没有给该字段赋值，则系统会自动给该字段赋值为 ‘‘汉’’。

```
create table tb_student2
(
  studentNo char(10) not null unique,
  studentName varchar(20) not null,
  sex char(2),
  birthday date,
  native varchar(20),
  nation varchar(30) default '汉',
  classNo char(6)
)engine = InnoDB;
```

需要注意的是，如果没有为字段指定默认值，MySQL 会自动为其分配一个，即如果字段可以取 null 值，这默认值就为 null，如果该字段被声明为 not null，那么默认值就取决于字段的类型：

- 对于没有声明 auto_increment 属性的数值型，默认值为 0；对于声明了 auto_increment 的则默认值为顺序中的下一个值。
- 对于除 timestamp 以外的日期和时间类型，默认值是该类型适当的 ‘‘零’’ 值；对于表中的第一个 timestamp 列，默认值是当前的日期和时间。

2. 存储引擎类型。存储引擎就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。因为在关系数据库中数据是以表的形式存储的，所以存储引擎简而言之就是指表的类型。数据库的存储引擎决定了表在计算机中的存储方式。使用 `show engines;` 可以查看系统支持的引擎类型和默认引擎，如下所示：

	Engine	Support	Comment	Transactions	XA	Savepoints
1	ARCHIVE	YES	Archive storage engine	NO	NO	NO
2	BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
3	MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
4	FEDERATED	NO	Federated MySQL storage engine	<null>	<null>	<null>
5	MyISAM	YES	MyISAM storage engine	NO	NO	NO
6	PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
7	InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
8	MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
9	CSV	YES	CSV storage engine	NO	NO	NO

其中，InnoDB 是系统默认的存储引擎，支持可靠的事务处理，是事务型数据库的首选。

5.2.3 查看表

查看表的名称：

```
show tables [{from | in} db_name];
```

例如查看数据库 db_school 中的所有表名：

```
use db_school;
show tables;
```

查看非当前数据库中的数据库名称时：

```
show tables from 基础数据;
-- 或
show tables in 基础数据;
```

查看表的基本结构：

这里的基本结构包括字段名、字段的数据类型、字段值是否允许为 null，是否有主键、是否有默认值等。

```
show columns {from | in} tb_name [ {from | in} db_name];
```

或者

```
{describe | desc} tb_name;
```

例如，查看 db_school 中表 tb_student2 的结构：

```
desc tb_student2;
-- 或
show columns from tb_student2 from db_school;
```

	Field	Type	Null	Key	Default	Extra
1	studentNo	char(10)	NO	PRI	<null>	
2	studentName	varchar(20)	NO		<null>	
3	sex	char(2)	YES		<null>	
4	birthday	date	YES		<null>	
5	native	varchar(20)	YES		汉	
6	classNo	char(6)	YES		<null>	

查看表的详细结构：

也就是查看创建表是 create table 的语句，语法格式为：

```
show create table tb_name;
```

例如查看 tb_student2 的详细结构：

```
show create table tb_student2;
```

	Table	Create Table
1	tb_student2	

```
CREATE TABLE `tb_student2` (
  `studentNo` char(10) NOT NULL,
  `studentName` varchar(20) NOT NULL,
  `sex` char(2) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `native` varchar(20) DEFAULT '汉',
  `classNo` char(6) DEFAULT NULL,
  UNIQUE KEY `studentNo` (`studentNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

5.2.4 修改表

5.2.4.1 添加字段

语法格式为:

```
alter table tb_name add [column] 新字段名 数据类型 [约束条件] [first | after
    已有字段名]
```

例如,向数据库 db_school 的表 tb_student2 中添加一个 int 型的字段 id,要求其不能为 null,取值唯一,自动增加,并将该字段添加为第一个字段。

```
use db_school;
alter table tb_student2 add id int not null unique auto_increment first;
```

再添加一个 varchar(16) 类型的字符串以描述学生所在的院系,要求设置默认值为信息学院,并将该字段添加到 nation 列之后。

```
alter table tb_student2 add department varchar(16) default '信息学院' after
    nation;
desc tb_student2;
```

	Field	Type	Null	Key	Default	Extra
1	id	int(11)	NO	UNI	<null>	auto_increment
2	studentNo	char(10)	NO	PRI	<null>	
3	studentName	varchar(20)	NO		<null>	
4	sex	char(2)	YES		<null>	
5	birthday	date	YES		<null>	
6	native	varchar(20)	YES		<null>	
7	nation	varchar(30)	YES		汉	
8	partment	varchar(16)	YES		信息学院	
9	classNo	char(6)	YES		<null>	

5.2.4.2 修改字段

alter table 语句提供了三个修改字段的子句,分别如下:

- change [column] 子句。该子句可以同时修改表中指定列的名称和数据类型。alter table 语句中可以同时添加多个 change [columns] 子句,只需要彼此之间使用逗号隔开。
- alter [column] 子句。该子句可以用来修改或删除表中指定列的默认值。

- modify [column] 子句。与 change [column] 子句不同，该子句只会修改指定列的数据类型，而不会干涉它的列名。另外，modify [column] 子句还可以通过 first 或 after 关键字修改指定列在表中的位置。上述三个子句的常用语法格式分别是：

```
alter table tb_name change [column] 原字段 新字段名 数据类型 [约束类型];
alter table tb_name alter [column] 字段名 [set | drop] default;
alter table tb_name modify [column] 字段名 数据类型 [约束条件][first|after 已
    有字段名];
```

例如，进行下面的操作：

```
alter table db_school.tb_student2
  change column birthday age tinyint null default 18;

alter table db_school.tb_student2
  alter column department drop default;

alter table db_school.tb_student2
  alter column department set default '经济学院';

alter table db_school.tb_student2
  modify column department varchar(20) not null after studentName;

desc tb_student2;
```

	Field	Type	Null	Key	Default	Extra
1	id	int(11)	NO	UNI	<null>	auto_increment
2	studentNo	char(10)	NO	PRI	<null>	
3	studentName	varchar(20)	NO		<null>	
4	department	varchar(20)	NO		经济学院	
5	sex	char(2)	YES		<null>	
6	age	tinyint(4)	YES		18	
7	native	varchar(20)	YES		<null>	
8	nation	varchar(30)	YES		汉	
9	classNo	char(6)	YES		<null>	

5.2.4.3 删除字段

语法格式为：

```
alter table tb_name drop [column] 字段名;
```

例如删除 id：

```
alter table db_school.tb_student2 drop column id;
desc db_school.tb_student2;
```

5.2.5 重命名表

1. rename [to] 子句

```
alter table db_school.tb_student2 rename to db_school.tb_student3;
```

2. rename table

```
rename table db_school.tb_student3 to db_school.tb_student2;
```

5.2.6 删除表

语法格式为：

```
drop table [if exists] 表1 [, 表2] ······;
```

语法说明如下：

- drop table 命令可以同时删除多个表，表和表之间用逗号隔开即可；
- if exists 用于在删除表之前判断要删除的表是否存在。例如删除 db_school 数据库中的所有表：

```
drop table db_school.tb_student2, db_school.tb_student;
```

注意，在删除表的同时，表的定义和表中的所有的数据都不会被删除，所以使用该语句需要格外小心。另外，用户在该表上的权限并不会自动被删除。

5.3 数据的完整性约束

关系模型的完整性规则就是对关系的某种约束条件。对关系模型施加完整性约束，则是为了在数据库应用中保障数据的正确性和一致性，防止数据库中存在不符合语义的、不正确的数据，这也是数据库服务器的功能之一。关系模型中有三类完整性约束：　　、　　　　和　　　　　　。其中**实体完整性和参照完整性是关系模型必须满足的完整性约束条件**，被称为　　　　　　　　。

5.3.1 实体完整性

实体完整性规则是指关系的主属性不能取空值，即主键和候选键在关系中所对应的属性不能取空值。MySQL 中实体完整性就是通过主键约束和候选键约束来实现的。

5.3.1.1 主键约束

主键是表中某一列或某些列所构成的一个组合。其中，由多个列组合而成的主键也被称为复合主键。主键的值必须是唯一的，而且构成主键的每一列的值都不允许为空。在 MySQL 中，主键列必须遵循如下一些规则：

1. 每个表中只能定义一个主键；
2. 主键的值，也被称为键值，必须能够唯一标识表中的每一行记录，且不能为 null。也就是说，表中的两条不同记录在主键上不能有相同的值。这就是　　　　　　　；

3. 复合主键不能包含不必要的列。也就是说,但从一个复合主键中删除一列后,如果剩下的列仍能满足唯一性原则,那么这个复合主键就是不争气的,这是 。
4. 一个列名在复合主键的列表中只能出现一次。

主键可以在 `create table` 或 `alter table` 语句中使用关键字 `primary key` 来实现。其方式有两种:

1. 一种是作为列级完整性约束，此时只需要在表中的某个字段的定义后加上关键字 `primary key`;
2. 另外一种是作为表级完整性约束,需要在表中所有字段的定义后添加一条 `primary key` 语法格式的子句。

例 5.1: 在数据库 db_school 中重新定义学生表,要求以列级完整性约束的方式定义主键。

```
use db_school;
create table tb_student
(
  studentNo char(10) primary key,
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6)
)engine = InnoDB;
```

例 5.2: 同上,但是以表级完整性约束的方式定义主键:

```
drop table db_school.tb_student;
use db_school;
create table tb_student
(
  studentNo char(10),
  studentName varchar(20),
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  primary key(studentNo)
)engine = InnoDB;
```

5.3.1.2 完整性约束的命名

与数据库中的表和视图一样,可以对完整性约束进行添加、删除和修改等操作。其中,为了删除和修改完整性约束,首先需要在定义约束的同时对其进行命名。命令完整性约束的方法是,在各种完整性约束的定义前加上关键字 `constraint` 和该约束的名字,其语法格式是:

```
constraint <symbol>
{primary key(主键字段列表)}
| unique(候选键字段列表)
| foreign key(外键字段列表) references tb_被参照关系(主键字段列表)
| check(约束条件表达式);
```

其中，symbol 为指定的约束名字，在完整性约束说明的前面被指定，其在数据库里必须是唯一的。倘若没有明确给出约束的名字，这 MySQL 会自动为其创建一个约束名字。

例如，指定上面的表中的主键约束的名称为 PK_student。

```
use db_school;
drop table tb_student;
create table db_school.tb_student
(
  studentNo char(10),
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  constraint PK_student primary key(studentNo)
) engine = InnoDB;
desc db_school.tb_student;
show columns from db_school.tb_students;
```

在定义完整性约束时，应当尽量地为其指定名字，以便在需要对完整性约束进行修改或删除操作时，可以更加容易地引用它们。需要注意的是，当前的 MySQL 版本只能给表级的完整性约束指定的名字，而无法给列级的完整性约束指定名字。因此表级完整性约束比列级完整性约束会更受欢迎。

5.3.1.3 候选键约束

与主键约束一样，候选键可以是表中的某一列，也可以是表中某些列所构成的一个组合。任何时候，候选键的值都必须是唯一的，且不能为 null。候选键可以在 create table 或 alter table 语句中使用关键字 unique 来定义，其实现方法与主键约束相似，同业有列级或者表级完整性约束两种形式。

例如：

```
use db_school;
create table tb_class
(
  classNo char(6) primary key,
  className varchar(20) not null unique,
  department varchar(30) not null,
  grade smallint,
```

```
  classNum tinyint
)engine = InnoDB;

# 如果将候选键定义为表级
drop table tb_class;
create table db_school.tb_class
(
  classNo char(6) primary key,
  className varchar(20) not null,
  department varchar(30) not null ,
  grade smallint,
  classNum tinyint,
  constraint UQ_class unique(className)
)engine = InnoDB;
```

MySQL 中 primary key 与 unique 之间存在如下几点区别：

- 一个表中只能创建一个 primary key，但是可以创建好几个 unique；
- 定义为 primary key 的列不允许有空值，但是定义为 unique 的字段允许空值；
- 定义为 primary key 约束时，系统会自动产生 primary key 索引，而定义 unique 时，系统自动产生 unique 索引。

5.3.2 定义参照完整性

在学校数据库的示例中，学生关系中的班级编号的取值需要参照班级关系中班级编号的取值。“ ” “ ”。外键是一个表中一个或一组属性，它不是这个表的主键，但它对应另一个表的主键。外键的主要作用是保证数据引用的完整性，保持数据的一致性。定义外键后，不允许删除外键引用的另外一个表中具有关联关系的记录。外键所属的表被称作参照关系，相关联的主键所在的表被称作被被参照关系。参照完整性规则定义的是外键和主键之间的引用规则，即外键的取值或者为空，或者等于被参照关系中某个主键的值。例如，学生关系中 ‘‘班级编号’’ 可以取如下两类值：

- 空值，表示还没来急的给该学生分班；
- 非空值，这是该值必须是班级关系中某个记录的班级编号字段值，表示该学生不可能被分陪到一个不存在的班级中。

在定义外键的时候，必须遵循下面的规则：

1. 被参照表必须已经使用 create table 创建，或者必须是当前正在创建的表。若是最后一种情形，则被参照表与参照表是同一个表，这样的表被称为自参照表，这种结构被称为自参照完整性。
2. 必须为被参照表定义主键或候选键。
3. 必须在被参照表的后面指定列名或列名的组合，这个列或列组合必须是被参照表的主键或候选键。

4. 尽管主键是不能包含空值的，但是允许外键中出现空值。这意味着，只要外键的某个非空值出现在出现在指定的主键中，这个外键的内容就是正确的。
5. 外键对应列的数目必须和被参照表的主键对应的列的数目相同。
6. 外键对应的列的数目必须和被参照表的主键对应的列的数据类型相同。

例如：

```
# 以表级完整性定义外键
use db_school;
drop table db_school.tb_student;
create table if not exists db_school.tb_student
(
  studentNo char(10),
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6) references db_school.tb_class(classNo)
)engine = InnoDB;

# 以表级完整性约束定义外键
drop table db_school.tb_student;
create table db_school.tb_student
(
  studentNo char(10),
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  constraint PK_student primary key(studentNo),
  constraint FK_student foreign key(classNo) references db_school.tb_class(
      classNo)
)engine = InnoDB;
```

在表 tb_student 的 classNo 列上定义外键约束之后，只有当某班级没有学生时才可以删除这个班级信息。MySQL 可以通过定义一个参照动作来修改这个规则，即定义外键时可以显式说明参照完整性约束的违约处理策略。

在给外键定义参照动作时，需要包括两部分：一是要指定参照动作适当的语句，即 update 和 delete 语句；二是需要指定采取的动作，即 cascade、restrict、set null、no action 和 set default，其中，restrict 为默认值。具体策略如下：

- restrict：限制策略，即当要删除或修改被参照表中被参照列上的其在外键中出现的值时，系统拒绝对被参照表的删除或修改操作。
- cascade：级联策略，即从被参照表中删除或修改记录时，自动删除或修改参照表中匹

配的记录。

- set null：置空策略，即当从被参照表中删除或修改记录的时候，设定参照表中与之对应的外键列的值为 null。这个策略需要被参照表中的外键列没有声明限定词 not null。
- no action：表示不采取实施策略，即当一个相关的外键值在被参照表中时，删除或修改被参照表中键值的动作不被允许。该策略的动作语义同 restrict。
- set default：默认值策略，即当从被参照表删除或修改记录行，设置参照表中与之对应的外键列的值为默认值。这个策略要求已经为该列定义了默认值。

例如：

```
drop table db_school.tb_student;
create table db_school.tb_student
(
  studentNo char(10),
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  constraint PK_student primary key(studentNo),
  constraint FK_student foreign key(classNo) references db_school.tb_class(
      classNo)
  on update restrict
  on delete cascade
)engine = InnoDB;
```

这里定义了两个参照动作，`on update restrict` 和 `on delete cascade`，`on update restrict` 表示当某个班级里有学生时不允许修改班级表中的该班级的编号；`on delete cascade` 表示当要删除班级表中的某个班级的编号时，如果该班级里有学生，就将相对应学生记录级联删除。

需要注意的是：

1. 外键只能引用主键和候选键。也就是说，只有当被参照关系的某个列上定义了主键或候选键，DBMS 才允许在参照关系的引用列上定义外键。
2. 外键只可以用在使用存储引擎 InnoDB 创建的表中，其它的存储引擎不支持外键。

5.3.3 用户定义的完整性

用户定义的完整性规则反映了某一具体应用所涉及的数据必须满足的语义要求。

关系模型提供定义和检验这类完整性规则的机制，其目的是用统一的方式由系统来处理它们，不再由应用程序来完成这项工作。在实际系统中，这类完整性规则一般在建立数据库表的同时进行定义，应用编程人员不需要在做考虑。如果某些约束条件在定义表时没有建立，则应用编程人员应该在各个模块的具体编程中通过应用程序进行检查和控制。

MySQL 支持集中用户自定义完整性约束，分别是非空检查约束、check 约束和触发器。其中触发器会在后面介绍，这里主要介绍前面两个约束。

1. 设置非空约束非空约束指某个字段的值不允许为空。对于使用里非空约束的字段，如果用户在添加数据时没有为其指定值，数据库系统就会报错。在 MySQL 中，非空约束的定义可以使用 create table 或 alter table 语句，在某个列后面添加 not null 作为限定词，来约束该值不能取空。
2. check 约束与非空约束一样，checku 约束也是在创建表或者修改表的同时，根据用户的实际完整性需要来定义的。check 约束需要指定限定条件，它可以分别定义为列级或表级完整性约束。列级 check 约束定义的是单个字段需要满足的要求，表级 check 约束可以定义表中多个字段之间应满足的条件。check 约束常用的语法格式为：

```
check (expr)
```

例如，在已有的数据库 db_school 中定义课程表 tb_course，要求自定义约束：每 16 个课时对应一个学分：

```
# check约束
use db_school;
create table tb_course
(
  courseNo char(6),
  courseName varchar(20) not null,
  credit int not null,
  courseHour int not null,
  term char(2),
  priorCourse char(6),
  constraint PK_course primary key (courseNo),
  constraint FK_course foreign key (priorCourse)
    references tb_course(courseNo),
  constraint CK_course check (credit = courseHour/16)
)engine = InnoDB;
```

再例如，在已有数据库 db_school 中定义成绩表 tb_course，要求成绩取值范围为 0～100。

```
use db_school;
create table tb_score
(
  studentNo char(10),
  courseNo char(6),
  score float check(score >= 0 and score <- 100),
  constraint PK_score primary key (studentNo, courseNo),
  constraint FK_score1 foreign key (studentNo)
    references db_school.tb_student(studentNo),
  constraint FK_score2 foreign key (courseNo)
    references db_school.tb_course(courseNo)
)engine = InnoDB;
```

5.3.4 更新表级完整性约束

5.3.4.1 删除约束

1. 删除外键约束如果外键约束是使用 constraint 子句命名的表级完整性约束，其语法格式为：

```
alter table <表名> drop foreign key <外键约束名>;
```

但要删除无命名的外键约束时，可先使用 `show create table` 语句查看系统给外键约束指定的名称，然后再删除该约束名。

例如，在表 tb_score 的字段 studentNo 上定义一个无命名的外键约束，然后删除它：

```
alter table tb_score
  add foreign key (studentNo)
    references db_school.tb_student(studentNo);
show create table tb_score;

alter table tb_score
    drop foreign key tb_score_ibfk_2;
```

2. 删除主键约束因为一个表只能定义一个主键，所以无论有没有给主键命名，均可以用 drop primary key 删除，其语法格式为：

```
alter table <表名> drop primary key;
```

例如：

```
alter table db_school.tb_student
    drop primary key;
```

3. 删除候选键约束删除候选键约束时，MySQL 实际删除的是唯一性索引，应使用 drop index 子句删除，如果没有给约束命令，MySQL 会自动将字段名定义为索引名。其语法格式为：

```
alter table <表名> drop index {约束名 | 候选字段名}
```

例如，如果没有给候选键命名，使用 drop index 子句删除的是定义候选键的字段名：

```
alter table tb_class
    drop index className;
```

如果使用了 constraint 子句给候选键命名，使用 drop index 子句删除的是约束名：

```
alter table db_school.tb_class
    drop index UQ_class;
```

5.3.4.2 添加约束

1. 添加主键约束：

```
alter table db_school.tb_student
    add constraint PK_student primary key (studentNo);
```

2. 添加外键约束:

```
alter table db_school.tb_score
    drop foreign key FK_score1;
alter table db_school.tb_score
    add constraint FK_score1 foreign key (studentNo)
  references db_school.tb_student(studentNo);
```

3. 添加候选键约束:

```
alter table db_school.tb_class
    add constraint UQ_class unique key(className);
```

注意,完整性约束不能直接被修改,若要修改某个约束,实际上是用 alter table 语句先删除它再增加一个同名的新约束。

5.4 练习题

这里面的这个语句比较陌生:`constraint CK_S check (CITY != 'London' or STATUS = 20)` 这句表示城市为 "London" 时其 STATUS 均为 20。

```
drop database db_sp;
create database if not exists db_sp
  default character set gb2312
  default collate gb2312_chinese_ci;
use db_sp;
create table S
(
  SNO char(10),
  SNAME varchar(10) unique not null,
  STATUS int,
  CITY varchar(10),
  constraint PK_S primary key (SNO),
  constraint CK_S check (CITY != 'London' or STATUS = 20)
)engine = InnoDB;

create table P
(
  PNO char(10),
  PNAME varchar(10) not null,
  COLOR varchar(10) check (COLOR in ('Red', 'Yellow', 'Green', 'Blue')),
  WEIGHT float,
  constraint PK_P primary key (PNO)
)engine = InnoDB;
```

```
create table SP
(
  SNO char(10),
  PNO char(10),
  QTY int,
  constraint PK_SP primary key (SNO, PNO),
  constraint FK_SP1 foreign key (SNO)
    references db_sp.S(SNO),
  constraint FK_SP2 foreign key (PNO)
    references db_sp.P(PNO)
)engine = InnoDB;
```

第 6 章 数据查询

在 MySQL 中，使用 select 语句能从数据表或者视图中查询满足条件的记录。select 语句的功能强大、使用灵活，其数学理论基础是关系数据模型中对表对象的一组关系运算，即选择 (selection)、投影 (projection) 和连接 (join)。

6.1 select 语句

使用 select 语句能够在需要时从数据库中快捷方便地检索、统计或输出数据。该语句的执行过程是从数据库中选取匹配的特定行和列，并将这些数据组织成一个结果集，然后以一张临时表的形式返回。在 MySQL 中，select 语句的语法内容较多，本书主要介绍一些简单的常用的语法。select 语法的常用语法格式为：

```
select [all | distinct | distinctrow] <目标列表达式1> [, <目标列表达式2>] ...
  from <表名1或视图名1> [, <表名2或视图名2>] ...
  [where <条件表达式>]
  [group by <列名1> [having <条件表达式>]]
  [order by <列名2> [asc | desc]]
  [limit [m,] n];
```

语法说明如下：

1. all | distinct | distinctrow：为可选项，用于指定是否返回结果集中的重复行。如果没有指定，默认为 `all`；如果指定了 distinct 或者 distinctrow，则会消除结果集中的重复行，这两个词是同一词，且这两个关键字应用于 select 语句中所指定的所有列，故在 select 语句中只需要指定一次，不需要在每个目标前都指定。
2. select 子句：用于指定要显示的字段或者表达式。from 子句用于指定数据来源于哪些表或视图；where 子句为可选项，用于指定对记录的过滤条件；group by 子句为可选项，用于将查询结果集按指定的字段值分组；having 子句为可选项，用于指定分组结果集的过滤条件；order by 子句为可选项，用于将查询结果集按指定字段值的升序或降序排列；limit 子句为可选项，用于指定查询结果集包含的记录数。

在 select 语句中，所有可选子句必须按照 select 语句的语法格式所罗列的顺序使用。表6.1总结了这些子句的顺序：

select 即可完成简单的单表查询，也可以实现复杂的连接查询和嵌套查询。

首先需要准备一些数据。为此我决定调整一下学习顺序，首先学一下数据插入。

6.2 一些数据准备工作

```
# 准备数据库
drop schema if exists db_sp;
```

表 6.1: select 子句的顺序

子句	说明	是否必须使用
select	指定返回的列或表达式	是
from	指定检索数据的表	仅在从表选择数据时使用
where	指定行级过滤条件	否
group by	指定分组字段	仅在按组计算计算聚合时使用
having	指定组级过滤条件	否
order by	指定排序字段和排序方式	否
limit	指定返回的记录数	否

```
drop schema if exists db_school;
create database db_school
  default character set gb2312
  default collate gb2312_chinese_ci;
create table db_school.tb_class
(
  classNo char(6) not null unique,
  className varchar(30) not null,
  department varchar(30) not null,
  grade int,
  classNum int,
  constraint PK_class primary key (classNo)
)engine = InnoDB;

create table db_school.tb_student
(
  studentNo char(10) not null unique,
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(30),
  nation varchar(20) default '汉',
  classNo char(6) not null,
  constraint PK_student primary key (studentNo),
  constraint FK_student foreign key (classNo)
    references db_school.tb_class (classNo)
)engine = InnoDB;

create table db_school.tb_course
(
  courseNo char(5) not null,
  courseName varchar(30) not null,
  credit tinyint not null,
  courseHour int not null,
  term tinyint not null,
  priorCourse char(5),
```

```
  constraint PK_course primary key (courseNo)
)engine = InnoDB;

create table db_school.tb_score
(
  studentNo char(10) not null,
  courseNo char(5) not null,
  score int check (score >= 0 and score <= 100),
  constraint FK_score1 foreign key (studentNo)
    references db_school.tb_student (studentNo),
  constraint FK_score2 foreign key (courseNo)
    references db_school.tb_course (courseNo)
)engine = InnoDB;

# 插入数据
insert into db_school.tb_class
    values
      ('AC1301', '会计13-1班', '会计学院', 2013, 35),
      ('AC1302', '会计13-2班', '会计学院', 2013, 35),
      ('CS1401', '计算机14-1班', '计算机学院', 2014, 35),
      ('IS1301', '信息系统13-1班', '信息学院', 2013, null),
      ('IS1401', '信息系统14-1班', '信息学院', null, 30);

insert into db_school.tb_student
    values
      ('2013110101', '张晓勇', '男', '1997-12-11', '山西', default, 'AC1301')
        ,
      ('2013110103', '王一敏', '女', '1996-03-25', '河北', default, 'AC1301')
        ,
      ('2013110201', '江山', '女', '1996-03-25', '内蒙古', '锡伯', 'AC1302'),
      ('2013110202', '李明', '男', '1996-01-14', '广西', '壮', 'AC1302'),
      ('2013310101', '黄菊', '女', '1995-09-30', '北京', default, 'IS1301'),
      ('2013310103', '吴昊', '男', '1995-11-08', '河北', default, 'IS1301'),
      ('2014210101', '刘涛', '男', '1997-04-03', '湖南', '侗', 'CS1401'),
      ('2014210102', '郭志坚', '男', '1997-02-21', '上海', default, 'CS1401')
        ,
      ('2014310101', '王林', '男', '1996-10-09', '河南', default, 'IS1401'),
      ('2014310102', '李怡然', '女', '1996-12-31', '辽宁', default, 'IS1401')
        ;

insert into db_school.tb_course
    values
      ('11003', '管理学', 2, 32, 2, null),
      ('11005', '会计学', 3, 48, 3, null),
      ('21001', '计算机基础', 3, 48, 1, null),
      ('21002', 'OFFICE高级应用', 3, 48, 2, '21001'),
      ('21004', '程序设计', 4, 64, 2, '21001'),
```

```
        ('21005', '数据库', 4, 64, 4, '21004'),
        ('21006', '操作系统', 4, 64, 5, '21001'),
        ('31001', '管理信息系统', 3, 48, 3, 21004),
        ('31002', '信息系统_分析与设计', 2, 32, 4, '31001'),
        ('31005', '项目管理', 3, 48, 5, '31001');

insert into db_school.tb_score
    values
        ('2013110101', '11003', 90),
        ('2013110101', '21001', 86),
        ('2013110103', '11003', 89),
        ('2013110103', '21001', 88),
        ('2013110201', '11003', 78),
        ('2013110201', '21001', 92),
        ('2013110202', '11003', 82),
        ('2013110202', '21001', 85),
        ('2013310101', '21004', 83),
        ('2013310101', '31002', 68),
        ('2013310103', '21004', 80),
        ('2013310103', '31002', 76),
        ('2014210101', '21002', 93),
        ('2014210101', '21004', 89),
        ('2014210102', '21002', 95),
        ('2014210102', '21004', 88),
        ('2014310101', '21001', 79),
        ('2014310101', '21004', 80),
        ('2014310102', '21001', 91),
        ('2014310102', '21004', 87);
```

6.3 单表查询

6.3.1 选择字段

1. 查询指定字段例如查询所在班级的班级编号、所属学院和班级名称

```
select classNo, department, className from db_school.tb_class;

#> AC1301    会计学院    会计13-1班
#> AC1302    会计学院    会计13-2班
#> CS1401    计算机学院   计算机14-1班
#> IS1301    信息学院    信息系统13-1班
#> IS1401    信息学院    信息系统14-1班
```

从班级表中查询出所有院系名称

```
select department from db_school.tb_class;
```

```
#> 会计学院
#> 会计学院
#> 计算机学院
#> 信息学院
#> 信息学院
```

如果想去掉上面的重复记录:

```
select distinct department from db_school.tb_class;

#> 会计学院
#> 计算机学院
#> 信息学院
```

2. 查询所有字段查询一个表中的所有字段有两种方法:一种是在 select 关键字后面列出所有的字段名;另一种是在 select 关键字后面直接使用 * 通配符。例如,查询全体学生的详细信息:

```
select * from db_school.tb_student;
```

3. 查询经过计算的值 select 子句的 "" 不仅可以是表中的字段名,也可以是表达式,还可以是字符串常量、函数等。例如,查询全体学生的姓名、性别和年龄。

```
select studentName, sex, 'Age:', year(now()) - year(birthday) from
    db_school.tb_student;

#> 张晓勇   男    Age:     21
#> 王一敏   女    Age:     22
#> 江山    女    Age:     22
#> 李明    男    Age:     22
#> 黄菊    女    Age:     23
#> 吴昊    男    Age:     23
#> 刘涛    男    Age:     21
#> 郭志坚   男    Age:     21
#> 王林    男    Age:     22
#> 李怡然   女    Age:     22
```

4. 定义字段的别名例如,查询全体学生的姓名、性别、年龄,要求给目标表达式取别名:

```
select
  studentName as 姓名,
  sex as 性别,
  year(now()) - year(birthday) as 年龄
from db_school.tb_student;

#> 姓名     性别   年龄
#> 张晓勇   男    21
#> 王一敏   女    22
#> 江山    女    22
```

表 6.2: where 子句常用的查询条件

查询条件	操作符
比较	=, <>, !=, <, <=, >, >=, !<, !>, not+ 含比较运算府的表达式
确定范围	between and, not between and
确定集合	in, not in
字符匹配	like, not like
空值	is null, is not null
多重条件	and, or

```
#> 李明    男    22
#> 黄菊    女    23
#> 吴昊    男    23
#> 刘涛    男    21
#> 郭志坚   男    21
#> 王林    男    22
#> 李怡然   女    22
```

需要注意的是：当自定义的别名中含有空格的时候，必须使用单引号将别名括起来。例如：

```
select
  studentName '姓  名',
  sex '性  别',
  year(now()) - year(birthday) '年  龄'
from db_school.tb_student;
```

6.3.2 选择指定记录

数据表中包含大量的数据，用户查询时可能只需要查询表中的指定数据，即对数据进行过滤。在 select 语句中，可以使用 where 子句，并根据 where 子句中指定的过滤条件（也称为搜索条件或查询条件），从 from 子句的中间结果中选择适当的数据行，实现数据的过滤。其语法格式为：

```
select 目标列表达式1, 目标列表达式2, ···, 目标列表达式n
from 表名
where 查询条件;
```

where 子句常用的查询条件如表6.2：

6.3.2.1 比较大小

比较运算符用于指定目标列表达式的值，当目标列表达式的值与指定的值相等时，返回 true，否则返回 false。例如，查询课时**大于等于 48 学时**的课程名称及学分：

```
# 查询少数民族学生的姓名、性别、籍贯和民族
select studentName, sex, native, nation
  from db_school.tb_student
```

```
  where nation != '汉';

# 或
select studentName, sex, native, nation
  from db_school.tb_student
  where not nation = '汉';

# 或
select studentName, sex, native, nation
  from db_school.tb_student
where nation <> '汉';
```

6.3.2.2 带 between· · ·and 关键字的范围查询

例如,查询出生日期在指定范围的:

```
select studentName, sex, birthday
  from db_school.tb_student
  where birthday between '1997-01-01' and '1997-12-31';

select studentName, sex, birthday
 from db_school.tb_student
where birthday not between '1997-01-01' and '1997-12-31';
```

6.3.2.3 带 in 关键字的集合查询

例如,查询籍贯是北京、天津和上海的学生的信息:

```
select *
from db_school.tb_student
where native in ('北京', '天津', '上海');

select *
from db_school.tb_student
where native not in ('北京', '天津', '上海');
```

6.3.2.4 带 like 关键字的字符串匹配查询

例如,查询学号为 ‘2013110201’ 的学生的详细情况:

```
select *
from db_school.tb_student
where studentNo like '2013110201';
```

查询所有姓王的学生的学号、姓名和班号

```
select studentNo, studentName, classNo
from db_school.tb_student
```

```
where studentName like '王%';
```

查询所有不姓王的学生的学号、姓名和班号

```
select studentNo, studentName, classNo
from db_school.tb_student
where studentName not like '王%';
```

查询姓名中包含' 林' 字的学生的学号、姓名和班号

```
select studentNo, studentName, classNo
from db_school.tb_student
where studentName like '%林%';
```

查询所有姓王且名字长度为三个中文字的学生的学号、姓名和班号

```
select studentNo, studentName, classNo
from db_school.tb_student
where studentName like '王__';
```

查询所有课程名称中包含下划线的课程信息

```
select *
from db_school.tb_course
where courseName like '%#_%' escape '#';
```

由于通常情况下，下划线是一个通配符，所以这里使用 # 作为转义字符，然后再使用 escape 指定一个转义字符 #。使用通配符的时候需要注意，MySQL 默认是不区分大小写的，如若需要区分大小写，需要更换字符集的校对规则；另外百分号不能匹配 null。

6.3.2.5 使用正则表达式

例如：查询课程名称中带有中文“系统”的课程信息

```
select *
from db_school.tb_course
where courseName regexp '系统';

# 或者
select *
from db_school.tb_course
where courseName like '%系统%';
```

查询课程名称中含有“管理”、“信息”或“系统”中文字符的所有课程信息

```
select *
from db_school.tb_course
where courseName regexp '管理|信息|系统';
```

6.3.2.6 带 is null 关键字的查询

例如,查询缺少和含有先行课的课程信息:

```
# 查询缺少先行课的课程信息
select *
from db_school.tb_course
where priorCourse is null;

# 查询所有含先行课的课程信息
select *
from db_school.tb_course
where priorCourse is not null;
```

注意,"is null" 不能用 "=null" 代替,"is not null" 不能用 "!= null" 代替。

6.3.2.7 带 and 或 or 的多条件查询

例如

```
# 查询学分>=3且学时大于32的课程名称、学分和学时数
select *
from db_school.tb_course
where credit >= 3 and courseHour > 32;

# 查询北京或上海的学生的姓名、籍贯和民族
select studentName, native, nation
from db_school.tb_student
where native = '北京' or native = '上海';

# 查询籍贯是北京或湖南的少数民族男生的姓名、籍贯和民族
select studentName, native, nation
from db_school.tb_student
where (native = '北京' or native = '湖南') and nation != '汉' and sex = '男';
```

6.3.3 对查询结果排序

```
# 查询学生的姓名、籍贯和民族并将查询结果按姓名升序排列
select studentName, native, nation
from db_school.tb_student
order by studentName;

# 查询学生选课成绩大于85分的学号、课程号和成绩信息，并将查询结果先按学号升序
    排列，再按成绩降序排列。
select *
from db_school.tb_score
where score > 85
order by studentNo, score desc;
```

注意，当对空值进行排序时，order by 会将该空值作为最小值对待。

6.3.4 限制查询结果的数量

limit 的用法格式为：

```
limit [位置偏移量, ] 行数
```

其中，行数是指定需要返回的记录数；“位置偏移量”是一个可选参数，指示 MySQL 从哪一行开始显示，第一条记录的位置偏移量为 0，第二条记录的位置偏移量为 1······ 以此类推。如果不指定位置偏移量，系统将会从表中的第一条记录开始显示。例如：

```
# 查询成绩排名在第3至第5的学生学号、课程号和成绩
select studentNo, courseNo, score
from db_school.tb_score
order by score desc
limit 2, 3;
```

从 MySQL5.0 开始，可以使用 limit 的另一种语法：

```
limit 行数 offset 位置偏移量
```

例如上面的案例代码可以修改成：

```
select studentNo, courseNo, score
from db_school.tb_score
order by score desc
limit 3 offset 2;
```

6.4 分组聚合查询

分组聚合查询是通过把聚合函数（如 count()、sum() 等）添加到一个带有 group by 分组子句的 select 语句中实现的。

6.4.1 使用聚合函数查询

聚合函数是 MySQL 提供的一类系统内置函数，常用于对一组值进行计算，然后返回单个值。使用聚合函数可以对数据进行分析。表6.3列出来常用的聚合函数：

其中，如果指定关键字 distinct，则表示在计算时要取消指定列中的重复值；如果不指定，默认不取消重复值。注意，除了 `count(*)` 外，其余的聚合函数，包括 count() 都会忽略空值。例如：

```
# 查询学生总人数
select count(*) from db_school.tb_student;

# 查询选修了课的学生总人数
use db_school;
```

表 6.3: 常用的聚合函数

函数名	说明
count([distinct | all] *)	统计数据表中的记录数
count([distinct | all])	统计数据表的一列中值的个数
max([distinct | all])	求数据表中一列值中的最大值
min([distinct | all])	求数据表中一列值中的最小值
sum([distinct | all])	求数据表中一列值的总和
avg([distinct | all])	求数据表中一列值的平均值

```
select count(distinct studentNo) from tb_score;

# 计算选修课程编号为21001的学生平均成绩
select avg(score) from tb_score
where courseNo = '21001';

# 计算选修课程编号为21001的学生最高分
select max(score) from tb_score
where courseNo = '21001';
```

6.4.2 分组聚合查询

在 select 语句中，允许使用 group by 子句对数据进行分组运算。分组运算的目的是为了细化聚合函数的作用对象。如果不对查询结果进行分组，聚合函数作用于整个查询结果，对查询结果分组后，聚合函数分别作用于每个组，查询结果就会按组聚合输出。group by 子句的语法格式是：

```
[group by 字段列表] [having <条件表达式>]
```

其中，group by 对查询结果按字段列表进行分组，字段值相等的记录分为一组；指定用于分组的字段可以是一列，也可以是多个列，彼此之间使用逗号分隔；having 短语对分组结果进行过滤，仅输出满足条件的组。

注意：使用 group by 子句后，select 子句的目标列表达式中只能包含 group by 子句中的字段列表和聚合函数。

例如：

```
# 查询各个课程号及相应的选课人数
select courseNo as '课程编号', count(studentNo) as '选课人数'
from tb_score
group by courseNo;
```

可以看出，使用 group by 的时候 select 子句中最好包含分组字段，否则，count() 的值四级意义不明确。对于 group by 的使用，需要注意下面几点：

1. group by 子句中列出的每个字段都必须是检索列或有效的表达式，但不能是聚合函数。如果在 select 语句中使用表达式，则必须在 group by 中使用相同的表达式，不能使用别名。

2. 除聚合函数外，select 子句中的每个列都必须在 group by 中给出。
3. 如果英语分组的列中含有 null 值，则 null 作为一个单独的分组返回；如果该列中存在多个 null 值，则将这些 null 值所在的行分为一组。

例如：

```
# 查询每个学生的选课门数、平均分、最高分
select studentNo, count(*) 选课门数, avg(score) 平均分, max(score) 最高分
from tb_score
group by studentNo;

# 查询平均分在80分以上的每个同学的选课门数、平均分和最高分
select studentNo, count(*) 选课门数, avg(score) 平均分, max(score) 最高分
from tb_score
group by studentNo
having avg(score) >= 80;

# 查询有2门课以上(含2门)课程成绩大雨88分的学生学号、学生姓名、及88分以上的课
    程数
select studentNo, count(*) 课程数
from tb_score
where score > 88
group by studentNo
having count(*) >= 2;

# 查询所有学生选课的平均成绩，但只有当平均成绩大于80的情况下才输出
select avg(score) 平均分
from tb_score
having avg(score) >= 80;
```

where 和 having 的主要区别在于作用对象不同：where 子句作用于基本表或视图，主要用于过滤基本表或视图中的数据行，从中选择满足条件的记录；having 短语作用于分组后的每个组，主要通过用于过滤分组，从中选择满足条件的组，即 having 短语作用于分组后的聚合值而不是特定行的值来过滤数据。

此外，having 短语中的条件还可以包含聚合函数，而 where 子句中则不可以；where 子句在数据分组钱进行过滤，having 短语则在数据分组后进行过滤。因而，where 子句排除的行不包含在分组中，这就有可能改变聚合值，从而影响 having 子句基于这些值过滤掉的分组。

如果一条 select 语句拥有一个 having 短语而没有 group by 子句，则会把表中的所有短语都聚合到一个分组中。

6.5 连接查询

前面介绍的查询都是针对一个表进行的。如果一个查询同时涉及两个或多个表，则称为连接查询。连接查询是关系数据库中重要的查询方式，其中包括交叉连接、内连接和外

连接。当两个或多个表中存在相同意义的字段时，便可以通过这些字段对相关的表连接查询。

6.5.1 交叉连接

交叉连接又称笛卡尔积，即把一张表的每一行与另一个表的每一行连接起来，返回两张表的每一行相连接后所有可能的搭配，其连接结果会产生一个没有意义的记录，所以这种查询实际很少使用。交叉连接所对应的 SQL 语句的语法结构为：

```
select * from 表1 cross join 表2;
-- 或
select * from 表1, 表2;
```

例如，查询学生表和成绩表的交叉连接：

```
select * from tb_student cross join tb_score;
```

交叉连接返回的查询结果集的记录数等于其所连接的两张表记录函数的乘积。因此，对于存在大量数据的表，应该避免使用交叉连接。同时，也可以在 from 子句的交叉连接后面，使用 where 子句过滤条件，减少返回的结果集。

6.5.2 内连接

内连接通过在查询中设置连接条件来移除交叉连接查询结果集中某些数据行。具体而言，内连接就是使用比较运算符进行表间某些字段值的比较操作，并将相匹配的数据行组成新的记录，其目的是为了消除交叉连接中某些没有意义的数据行。也就是说，内连接查询中，只有满足条件的记录才会出现在结果集中。内连接有两种表示形式：

- 使用 inner join 的显式语法结构：

```
select 目标列表达式1, 目标列表达式2, ······, 目标列表达式n
from table1 [inner] join table2
on 连接条件
[where 过滤条件];
```

- 使用 where 子句定义连接条件的隐式语法结构：

```
select 目标列表达式1, 目标列表达式2, ······, 目标列表达式n
from table1 table2
where 连接条件 [and 过滤条件];
```

上述两种表示形式的差别在于：使用 inner join 连接后，from 子句中的 on 子句可用来设置连接表的连接条件，而其他过滤条件则可以在 select 语句中的 where 子句中指定；而使用 where 子句定义连接条件的形式，表于表之间的连接条件和查询时的过滤条件均在 where 子句中指定。

6.5.2.1 等值与非等值连接

例如:

```
# 查询每个学生选修课程的情况
use db_school;
select tb_student.*, tb_score.*
from tb_student, tb_score
where tb_student.studentNo = tb_score.studentNo;

# 查询会计学院全体同学的学号、姓名、籍贯、班级编号和所在班级名称
select studentNo, studentName, native, tb_student.classNo, className
from tb_student, tb_class
where tb_student.classNo = tb_class.classNo and department = '会计学院';

# 查询选修了课程"程序设计"的学生学号，姓名和成绩
select a.studentNo, studentName, score
from tb_student as a, tb_course as b, tb_score as c
where a.studentNo = c.studentNo and b.courseNo = c.courseNo
  and courseName = '程序设计';

-- 或
select a.studentNo, studentName, score
from tb_student as a join tb_course as b join tb_score as c
on a.studentNo = c.studentNo and b.courseNo = c.courseNo
where b.courseName = '程序设计';
```

6.5.2.2 自连接

若某个表与自身进行连接，称为自表连接或自身连接，简称自连接。使用自连接时，需要为表指定不同的别名，且对所有查询字段的应用都必须使用别别名限定，否则 select 操作会失败。例如:

```
# 查询与"数据库"这门课学分相同的课程信息
select c1.*
from tb_course as c1, tb_course as c2
where c1.credit = c2.credit and c2.courseName = '数据库';

-- 或
select c1.*
from tb_course c1 join tb_course c2
on c1.credit = c2.credit
where c2.courseName = '数据库';
```

6.5.3 自然连接

自然连接只有当连接字段在两张表中的字段名都相同时才使用，否则返回的时笛卡尔积的结果。自然连接在 from 语句中使用关键字 natural join。例如：

```
# 查询每个学生及其选修课程的情况，要求显示学生学号、姓名、选修课程号和成绩
select a.studentNo, studentName, courseNo, score
from tb_student as a natural join tb_score as b;
```

6.5.4 外连接

连接查询是要查询多个表中相关联的行，内连接查询只返回查询结果集中符合查询条件（即过滤条件）和连接条件的行。但是有时候查询结果也需要显示不满足连接条件的记录，即返回查询结果集不仅仅包括符合连接条件的行，还包括两个两个连接表中不符合连接条件的行。

外连接首先将连接的两张表分为基表和参考表，然后再以基表为依据返回满足和不满足连接条件的记录，就好像在参考表中增加了一条全部由空值组成的万能行，它可以和基表中所有不满足连接条件的记录行进行连接。

外连接按照连接表的顺序可以分为左外连接和右外连接两种。

6.5.4.1 左外连接

左外连接，也被称为左连接，用于返回关键字左边表（基表）的所有记录，并用这些记录与该关键字右边表 (参考表) 中的记录进行匹配，如果左表的某些记录在右表中没有匹配的记录，就和右表的万能行连接，即右表对应的字段值均被设置为空值 null。

例如：

```
# 使用左外连接查询所有学生及其选修课程的情况，包括没有选修课程的学生，
# 要求显示学号、姓名、性别、班号、选修课程号和成绩
# 首先在学生表中插入一条记录
insert into tb_student
    values
      ('2013310102', '林海', '男', '1996-01-18', '北京', '满', 'IS1301');
# 然后进行左连接查询
select a.studentNo, studentName, sex, classNo, courseNo, score
from tb_student as a left join tb_score as b
on a.studentNo = b.studentNo;
```

6.5.4.2 右外连接

也称为右连接，以右表为基表。例如，使用右外连接查询所有学生及其选修课程的情况，包括没有选修课程的学生，要求显示学号、姓名、性别、班号、选修的课程号和成绩。

```
select courseNo, score, b.studentNo, studentName, sex, classNo
from tb_score as a right join tb_student as b
```

```
on a.studentNo = a.studentNo;
```

6.6 子查询

子查询也被称为嵌套查询，是将一个查询语句嵌套在另一个查询语句的 where 子句或 having 短语中，前者被称为内层查询或子查询，后者被称为外层查询或父查询。在整个 select 语句中，先计算子查询，然后将子查询的结果作为父查询的过滤条件。嵌套查询可以用多个简单查询构成一个复杂的查询，从而增加 SQL 的查询能力。

6.6.1 带 in 的子查询

```
# 查询选修了课的学生
select studentName
from tb_student
where tb_student.studentNo in
      (select distinct tb_score.studentNo from tb_score);

# 或者不使用子查询
select distinct studentName
from tb_student, tb_score
where tb_student.studentNo = tb_score.studentNo;

# 查询没有选修课的学生
select studentName from tb_student
where tb_student.studentNo not in
      (select distinct tb_score.studentNo from tb_score);
```

6.6.2 带比较运算符的子查询

例如，查询班级 “计算机 14-1 班” 所有学生的学号、姓名：

```
# 查询班级“计算机14-1班”所有学生的学号、姓名
use db_school;
select studentNo, studentName from tb_student
where classNo =
      (select classNo from tb_class
      where className = '计算机14-1班');

# 或者使用连接查询
select studentNo, studentName
from tb_student left join tb_class
  on tb_student.classNo = tb_class.classNo
where className = '计算机14-1班';
```

查询和 “李明” 在同一个班级学习的学生学号、姓名和班号：

```sql
select studentNo, studentName, classNo from tb_student as s1
where classNo =
      (select classNo from tb_student as s2
      where s2.studentName = '李明') and s1.studentName != '李明';

# 或者使用连接查询
select s2.studentNo, s2.studentName, s2.classNo
from tb_student as s1, tb_student as s2
where s1.classNo = s2.classNo and s1.studentName = '李明'
having s2.studentName != '李明';
```

查询男生中比某个女生出生年份晚的学生姓名和出生年份。

```sql
select studentName as 姓名, year(birthday) as 出生年份 from tb_student
where sex = '男' and year(birthday) > any(
  select year(birthday) from tb_student where sex = '女'
);
```

查询男生中比某个女生出生年份晚的学生姓名和出生年份

```sql
select studentName as 姓名, year(birthday) as 出生年份
from tb_student
where sex = '男' and year(birthday) > all(
  select year(birthday) from tb_student where sex = '女'
);

# 或者使用聚合函数
select studentName as 姓名, year(birthday) as 出生年份
from tb_student
where sex = '男' and year(birthday) > (
  select max(year(birthday)) from tb_student where sex = '女'
)
```

6.6.3 带 exists 关键字的子查询

使用关键字 exists 构造子查询时，系统会对子查询进行运算以判断它是否返回结果集。如果子查询的结果集不为空，则 exists 返回的结果为 true，此时外层查询语句将进行查询；如果子查询的结果集为空，则 exists 返回的结果为 false，此时外层查询语句将不再进行查询。由于带 exists 的子查询只返回 true 或 false，内层查询的 select 子句给出字段名没有实际意义，所以其目标列表达式通常用 *。例如，查询选修了课程号为 “31002” 的学生的姓名：

```sql
select studentName from tb_student as a
where exists
(select * from tb_score as b
where a.studentNo = b.studentNo and courseNo = '31002');
```

查询没有选修课程号号为 “31002” 的学生姓名:

```
select studentName from tb_student as a
where not exists(
    select *
    from tb_score as b
  where a.studentNo = b.studentNo and courseNo = '31002'
);
# 或者使用not in
select studentName
from tb_student
where studentNo not in (
  select studentNo from tb_score
  where courseNo = '31002'
);
```

查询选修了所有课程的学生姓名:

```
# 也就是说，查询这样的学生，没有一门课程是他没有选修的（双重否定）
select studentName
from tb_student as x
where not exists(
  select *
  from tb_course as c
  where not exists(
      select *
      from tb_score
    where x.studentNo = x.studentNo
    and c.courseNo = c.courseNo
  )
);
```

6.7 联合查询

使用 union 关键字可以把来自多个 select 语句的结果组合到一个结果集中，这种查询方式称为并运算或者联合查询。合并时，多个 select 子句中对应的字段数和数据类型必须相同。其语法格式是:

```
select -from -where
union [all]
select -from -where
[... union [all]]
select -from -where
```

其中不使用关键字 all，执行的时候去掉重复的记录，所有返回的行都是唯一的；使用关键字 all 的作用是不去掉重复的记录，也不对结果进行自动排序。例如，使用 union 查询 “管理学” 或 “计算机基础” 的学生学号:

```
select studentNo
from tb_score, tb_course
where tb_score.courseNo = tb_course.courseNo and courseName = '管理学'
union
select studentNo
from tb_score, tb_course
where tb_score.courseNo = tb_course.courseNo and courseName = '计算机基础';
# 或者
select distinct studentNo
from tb_score, tb_course
where tb_course.courseNo = tb_score.courseNo
and (courseName = '管理学' or courseName = '计算机基础');
```

使用 union 查询选修了 ‘‘管理学’’ 或 ‘‘计算机基础’’ 的学生学号:

```
select studentNo
from tb_score, tb_course
where tb_score.courseNo = tb_course.courseNo and courseName = '管理学'
union all
select studentNo
from tb_score, tb_course
where tb_score.courseNo = tb_course.courseNo and courseName = '计算机基础';

# 或者
select studentNo
from tb_score, tb_course
where tb_course.courseNo = tb_score.courseNo
and (courseName = '管理学' or courseName = '计算机基础');
```

使用 union 查询的时候需要注意以下几点:

1. union 语句必须由两条或两条以上的 select 语句组成,且彼此之间使用关键字 union 分隔;
2. union 语句中的每个 select 子句必须包含相同的列、表达式或聚合函数;
3. 每个 select 子句中对应的目标列的数据类型必须兼容。目标列的数据类型不必完全相同,但必须是 MySQL 可以隐含转换的类型,例如,不同的数值类型或不同的日期函数。
4. 第一个 select 子句中的目标列名会被作为结果集的列名;
5. 联合查询中只能使用一条 order by 子句或 limit 子句,且它们必须置于最后一条 select 子句之后;

使用联合查询是标准 SQL 直接支持的集合操作,相当于集合操作中的并运算。MySQL 的当前版本只支持并运算,交运算和差运算只能用子查询来实现。

查询选修了计算机基础和管理学的学生学号:

```
select studentNo
from tb_score, tb_course
```

```
where tb_score.courseNo = tb_course.courseNo and courseName = '计算机基础'
and studentNo in (
  select studentNo from tb_score, tb_course
  where tb_score.courseNo = tb_course.courseNo and courseName = '管理学'
);
```

查询选修了计算机基础但没有管理学的学生学号:

```
select studentNo
from tb_score, tb_course
where tb_score.courseNo = tb_course.courseNo and courseName = '计算机基础'
and studentNo not in (
  select  studentNo from tb_score, tb_course
  where tb_score.courseNo = tb_course.courseNo and courseName = '管理学'
);
```

第 7 章 数据更新

MySQL 中提供了功能丰富的数据库更新操作语句，包括向数据库中插入数据的 insert 语句、修改数据的 update 语句，以及当数据不再使用时删除数据的 delete 语句。

7.1 插入数据

7.1.1 插入完整的数据记录

例如，向 tb_student 中插入一条新纪录：

```
use db_school;
insert into tb_student
    values
      ('2014210103', '王玲', '女', '1998-02-21', '安徽', '汉', 'CS1401');

# 指定字段添加
insert into tb_student(studentNo, studentName, sex, birthday, native, nation,
     classNo)
    values
      ('2013110102', '赵婷婷', '女', '1996-11-30', '天津', '汉', 'AC1301');

# 指定部分字段添加
insert into tb_student(studentNo, studentName, native, nation, sex, birthday,
     classNo)
    values
      ('2013110203', '孟颖', '上海', '汉', '女', '1997-03-20', 'AC1302');
```

7.1.2 为表的指定字段插入数据

为表的指定字段插入数据就是在 insert 语句中只给部分字段的插入值，而其他字段的值为缺省值，没有定义缺省值的字段应运行取空值。例如：

```
insert into tb_student(studentNo, studentName, sex, nation, classNo)
    values
      ('2014310103', '孙新', '男', '傣', 'IS1401');

select * from tb_student;
```

7.1.3 同时插入多条数据记录

例如：

```
insert into tb_student(studentNo, studentName, sex, classNo)
    values
      ('2014310104', '陈卓卓', '女', 'IS1401'),
      ('2014310105', '马丽', '女', 'IS1401'),
      ('2014310106', '许江', '男', 'IS1401');
```

7.1.4 插入查询结果

该用法的语法格式为:

```
insert into tb_name1 (column_list1)
  select column_list2 from tb_name2 where (condition);
```

例如,假设要为表 tb_student 制作一个备份表 tb_student_copy,两个表的结构完全一致,现使用 insert· · ·select 将表 tb_student 中的数据备份到新建表 tb_student_copy 中:

```
create table tb_student_copy
(
  studentNo char(10),
  studentName varchar(20),
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10),
  classNo char(6),
  constraint PK_student primary key (studentNo)
);
insert into tb_student_copy
    select * from tb_student;
```

7.1.5 使用 replace 语句插入表数据

若一个待插入的表中存在有 primary key 或 unique 约束,而待插入的数据行中包含有与待插入表的已有数据行中相同的 primary key 或 unique 列值,那么 insert 语句将无法插入此行。此时,如果需要插入这行数据,则可使用 replace 语句来实现。例如:

```
replace into tb_student_copy(studentNo, studentName, sex, birthday, native,
    nation, classNo)
    values
      ('2013110101', '周旭', '男', '1996-10-01', '湖南', '汉', 'AC1301');
```

注意,如果数据表的某个字段定义了外码,使用 replace into 插入数据时依然会出错。

7.2 修改数据记录

其语法格式为:

```
update tb_name
  set column1=value1, ······, column n = value n
  [where <condition>];
```

7.2.1 修改特定数据记录

例如：

```
update db_school.tb_student
    set studentName = '黄涛',
      native = '湖北',
      nation = default
where studentNo = '2014210101';
```

7.2.2 修改所有数据记录

例如将成绩表 tb_score 中所有学生的成绩提高 5%：

```
update db_school.tb_score
    set score = score * 1.05;
```

7.2.3 带子查询的修改

例如，将选修了 ‘‘程序设计’’ 这门课的学生成绩置 0：

```
# 将选修了“程序设计”这门课的学生成绩置0
update db_school.tb_score
    set score = 0
where courseNo = (select courseNo from tb_course where courseName = '程序设计
    ');

# 查看修改结果
select studentNo, tb_score.courseNo, courseName, score
from tb_course, tb_score
where tb_course.courseNo = tb_score.courseNo and courseName = '程序设计';
```

7.3 删除数据记录

7.3.1 删除特定的数据记录

例如，删除王一敏的学生信息：

```
# 先看一下她的信息
select *
from tb_student_copy where studentName = '王一敏';
# 删除
```

```
delete from tb_student_copy
where studentName = '王一敏';
# 再看一下
select *
from tb_student_copy where studentName = '王一敏';
```

7.3.2 带子查询的删除

例如，将程序设计这门课的所有选课记录删除：

```
# 带子查询的删除
delete from db_school.tb_score
where courseNo = (select courseNo from tb_course where courseName = '程序设计
    ');
# 查看结果
select studentNo, tb_score.courseNo, courseName, score
from tb_course, tb_score
where tb_course.courseNo = tb_score.courseNo and courseName = '程序设计';
```

7.3.3 删除所有数据记录

例如，删除所有学生的选课记录

```
delete from db_school.tb_score;
# 查看结果
select *
from tb_score;
```

如果要删除表中的所有记录，还可以使用 truncate 语句。truncate 语句将直接删除原来的表并重新创建一个表，而不是逐行删除表中的记录，因此执行速度会比 delete 快。

```
# 或者, 使用truncate语句
truncate db_school.tb_student_copy;
```

truncate 语句使用起来非常简单，但是需要注意以下几点：

1. 由于 truncate 语句在功能上与不带 where 子句的 delete 语句相同，它将删除表中的所有数据，且无法恢复，因此使用的时候要非常谨慎。
2. 使用 truncate 语句后，表中的 auto_increment 计数器会被重新设置为该列的初始值。

第 8 章 索引

索引是数据库技术的一个重要概念与技术，也是 MySQL 的一个数据库对象。对于任何 DBMS，索引都是查询优化的最主要方式。当数据量非常大的时候，若没有合适的索引，数据库的查询性能会急剧下降。因此，建立索引的目的就是加快数据库检索的速度。

8.1 索引概述

对数据库中数据表进行查询操作时，系统对表中的数据主要有两种搜索扫描方式：一种是全表扫描、检索，另一种是利用数据表上建立的索引进行扫描。

全表扫描是将表中的所有数据记录从头至尾逐行读取，与查询结果进行对比，返回满足条件的记录。这种搜索方式需要读取相关表中的所有数据，需要进行大量的磁盘读写操作，当表中的数据量巨大的时候，查询效率会非常低。

索引访问这是通过搜索索引值，再根据索引值与记录的关系直接访问数据表中的记录行。例如，对学生表的姓名字段建立索引，即按照表中的姓名字段的数据进行索引，并为其建立指向数据表中记录所在位置的指针。

通过索引进行数据检索的方式类似与在图书馆查找图书的过程，即在图书馆首先通过检索书名或书号等信息，获得所需图书的位置信息，然后按照该位置信息直接到确定的书库、书架上拿取图书，而不是对图书馆的所有书籍进行逐库、逐书架的查找。因此，在数据库中建立适当的索引能有效提高检索的效率。

根据用途，MySQL 中的索引主要分为普通索引、唯一性索引、主键索引、聚簇索引及全文索引几类。

1. 普通索引普通索引（index）是最基本的索引类型。普通的索引列值可以取空值或重复值。创建普通索引时，通常使用的关键字是 index 或 key。
2. 唯一性索引唯一性索引（unique）与普通索引基本相同，区别仅是索引列值不能重复，但可以为空值。创建唯一性索引所使用的关键字是 unique。
3. 主键索引在 MySQL 中建立主键时，系统会自动创建主键索引。主键索引是一种唯一性索引。与唯一性索引不同在于其索引值不能为空。创建主键时，必须使用关键字 primary key。一般是在创建表的时候指定主键，也可以通过修改表的方式创建主键索引。每个表只能有一个主键。
4. 聚簇索引聚簇索引的索引顺序就是数据存储的物理存储数据，这样能保证索引值相近的元组所存储的物理位置也相近。对于非聚簇索引，索引顺序与数据的物理排列顺序无关。一个表只能又一个聚簇索引。并非所有的 MySQL 存储引擎都支持聚簇索引，目前只有 solidDB 和 InnoDB 支持。
5. 全文索引（fulltext）全文索引只能创建在数据类型为 varchar 或 text 的列上。建立全文索引后，能够在建立了全文索引的列上进行全文查找。全文索引只能在 MyISAM

存储引擎的表中创建。

在实际使用中，索引可以建立在单一列上，称为单列索引。也可以建立在多个列上，称为组合索引。

1. 单列索引：单列索引就是一个索引只包含原表中的一个列。一个表上可以建立多个单列索引。例如，建立在学生表中关于学号的索引，还可以建立关于姓名的索引、所在班级的索引等。
2. 组合索引：组合索引也称为复合索引或者多列索引。组合索引是指在表的多个列上创建一个索引。例如，在班级表 tb_class 的 ''所属院系'' 和 ''年级'' 两列上建立一个索引，即为组合索引。这个索引的含义是先按所属院系排序，若所在院系相同，则按照年级排序。这就是 。所谓的左前缀法则就是指先按照第一列（顺序排列在最左侧的字段）进行排序，当第一列的值相同的情况下再对第二列排序，依此类推。

8.2 查看数据表上所建立的索引

在 MySQL 中，使用 show index 语句能够查看数据表中是否建立了索引，以及所建立索引的类型及相关参数，其语法格式为：

```
show {index | indexs | keys} {from | in} tb_name [{from | in}] db_name;
```

该语句的功能是显示出表名为 tbl_name 的表上所有定义的索引名及索引类型。例如：

```
show index from db_school.tb_score;
```

上面的结果会是一个很长的表，如果是在终端运行，可以使用\G 参数：

```
show index from db_school.tb_score \G;
-- 结果
*************************** 1. row ***************************
        Table: tb_score
   Non_unique: 1
     Key_name: FK_score1
 Seq_in_index: 1
  Column_name: studentNo
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
*************************** 2. row ***************************
        Table: tb_score
   Non_unique: 1
```

```
      Key_name: FK_score2
  Seq_in_index: 1
   Column_name: courseNo
     Collation: A
   Cardinality: 0
      Sub_part: NULL
        Packed: NULL
          Null:
    Index_type: BTREE
       Comment:
 Index_comment:
       Visible: YES
2 rows in set (0.00 sec)
```

1. Table:索引所在表的名称;
2. Non_unique:索引是否为唯一性索引,如果为唯一性索引,该值为 0,如果不是,该值为 1。
3. Key_name:索引的名称;
4. Column_name:建立索引的列名称;
5. Collation: 说明以何种顺序（升序或降序）索引。如果是升序，取 A，如果显示为 NULL,表示无分类。

8.3 创建索引

MySQL 提供了三种创建索引的方法。一种是在创建表的同时创建索引,另外两种分别是在已存在的表上使用 create index 语句创建索引,或使用 alter table 语句添加索引。### 使用 create table 语句创建索引语法格式为:

```
create table tbl_name [col_name data_type]
  [constraint index_name] [unique] [index | key]
  [index_name] (index_col_name[length]) [asc | desc]
```

语法说明如下:

1. tbl_name:指定需要建立索引的表名。
2. index_name:指定所建立的索引名称。一个表中可以建立多个索引,而每个索引的名称必须是唯一的。
3. unique:可选项,指定所创建的是唯一性索引。
4. index_col_name: 指定要创建索引的列名。通常可考虑将查询语句在 where 子句和 join 子句里出现的列作为索引列。
5. length:可选项,用于指定使用列的前 length 个字符创建索引。使用列值的一部分创建索引有利于减小索引文件的大小,节省磁盘空间。当然,在某些情况下,只能对列的前缀进行索引。例如,由于索引列的长度有一个最大上限,所以如果索引列的长度超过的超过了这个上限,那么此时就需要利用前缀进行索引。另外,为 blob 或 text 类

型的数据列建立索引时必须使用前缀索引，前缀最长为 255 字节，但对于 MyISAM 和 InnoDB 表，前缀最长为 1000 字节。

6. asc | desc：可选项，指定索引是按升序还是降序排列，默认为升序。

例如，创建新表的同时创建普通索引：

```
create table tb_student1
(
  studentNo char(10) not null,
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  index(studentName)
);
```

创建新表时，建立唯一性索引：

```
create table tb_student2
(
  studentNo char(10) not null unique,
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6)
)
```

创建新表的同时建立主键索引：

```
create table tb_score1
(
  studentNo char(10),
  courseNo char(5),
  score float,
  constraint PK_score primary key (studentNo, courseNo),
  constraint FK_score11 foreign key (studentNo) references tb_student(
      studentNo),
  constraint FK_score12 foreign key (courseNo) references tb_course(courseNo)
);
-- 查看索引（在终端运行）
mysql> show index from db_school.tb_score1\G;
*************************** 1. row ***************************
        Table: tb_score1
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
```

```
 Column_name: studentNo
   Collation: A
 Cardinality: 0
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
Index_comment:
     Visible: YES
*************************** 2. row ***************************
       Table: tb_score1
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 2
 Column_name: courseNo
   Collation: A
 Cardinality: 0
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
Index_comment:
     Visible: YES
*************************** 3. row ***************************
       Table: tb_score1
  Non_unique: 1
    Key_name: FK_score12
Seq_in_index: 1
 Column_name: courseNo
   Collation: A
 Cardinality: 0
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
Index_comment:
     Visible: YES
3 rows in set (0.02 sec)
```

凡是被定义为主键或外键的序列，系统俊晖自动建立相应字段的索引。需要提醒的是，并非所有的数据库管理系统都会自动对主键、外键建立索引。在使用其他数据库管理系统的时候，应该查看手册或测试以下。

8.3.1 使用 create index 语句创建索引

用 create index 语句能够在一个已存在的表上创建索引,其语法格式是:

```
create [unique] index index_name
  on tbl_name (col_name [(length) ] [asc | desc], ...);
```

创建普通索引:

```
create index index_stu on db_school.tb_student(studentNo);
# 查看
show index from db_school.tb_student;
```

创建基于字段值前缀字符的索引:

```
# 在数据库db_school中课程表tb_score上建立一个索引,
# 要求按照课程名称courseName字段值前三个字符建立降序索引
create index index_course on db_school.tb_course(courseName(3) desc);
```

对于字符类型的排序,若是英文,按照字母序排序;若是中文,不同的系统有不同的处理规则。MySQL 是按照汉语拼音对应的英文字母顺序进行排序的。

创建组合索引:

```
# 在数据库db_school中表tb_book上建立图书类别(升序)和书名(降序)的组合索
    引, 索引名称为index_book
create table db_school.tb_book
(
  bclassNo char(10) unique not null,
  bookName varchar(20) not null
);
create index index_book on db_school.tb_book(bclassNo, bookname desc);
show index from db_school.tb_book;
```

8.3.2 使用 alter table 语句创建索引

在 MySQL 中,除了使用 create index 语句创建一个已存在的表上创建索引之外,还可以使用 alter table 语句实现类似的功能。语法格式如下:

```
alter table tbl_name add [unique | fulltext]
[index | key] [index_name] (col_name [length] [asc | desc], ...);
```

使用 alter table 创建普通索引

```
alter table db_school.tb_student1
    add index idx_studentName(studentName);
```

除了使用 alter table 语句的 add index 建立索引,使用 add constraint 子句能够在表中添加主键或外键约束,与此同时,系统会自动创建相应的索引。

8.4 删除索引

在 MySQL 中,使用 drop index 语句或 alter table 语句能够删除一个不再需要的索引。

1. 使用 drop index 语句删除索引例如:

```
drop index idx_studentName on db_school.tb_student1;
```

1. 使用 alter table 语句删除索引例如:

```
alter table db_school.tb_student drop index index_stu;
```

使用 alter table 语句的 drop constraint 子句能够在表中删除主键或外键约束,同时也就删除了相应于主键、外键的索引。

注意:

1. 若删除表中的某一列,而该列是索引项,则该列也会从索引中被删除;
2. 如果组成索引的所有列都被删除,则整个索引将会被删除。

8.5 对索引的进一步说明

8.5.1 使用索引时的问题

尽管使用索引可以大大加快查询响应的速度,提高 MySQL 的检索性能,但是过多使用索引将会影响系统的性能,其主要原因如下:

1. 降低更新表中数据的速度索引在提高查询速度的同时,会降低更新数据表的速度。在更新数据表时,系统会自动地更新索引列的数据,以确保索引与表中的数据保持一致,这可能需要重新组织一个索引。如果表上建立的索引很多,会非常浪费更新操作的时间,由此降低 insert、update、delete 及其他写入操作的效率。表中建立的索引越多,则更新表的时间越长。
2. 增加存储空间无论是采取 InnoDB 还是 MyISAM 存储引擎,索引都需要占用磁盘空间。如果有大量的索引,索引文件可能会比数据文件更快地达到最大的文件尺寸。特别是如果在一个大表上创建了多种组合索引,索引文件的大小会膨胀得非常快。

因此,索引只是提高检索效率的一个手段。如果 MySQL 数据库中存在大量数据表,而这些表的更新操作比较多,则需要认真分析,设计有效的索引,或者优化所使用的查询语句。

8.5.2 使用索引的建议

1. 使用索引有利于提高数据查询的效率,但是会影响数据更新的速度。不恰当的索引会降低系统性能。因此,在插入、修改、删除操作较多的数据表上避免建立过多的索引。
2. 数据量较小的数据表最好不要建立索引。
3. 使用组合索引时,严格遵循左前缀法则。

4. 在查询表达式中经常使用、有较多不同值的字段上建立索引。避免在不同值较少的字段上建立索引。例如，‘‘性别’’ 字段的值只有两个：‘‘男’’ 和 ‘‘女’’。在这类字段上建立索引不仅不会提高查询效率，反而会降低更新速度。
5. where 子句中尽量避免将索引列作为表达式的一部分。在使用 like 时，避免开头使用通配符。例如，“like %aaa%’’ 会使索引失效，而 “like aaa%’’ 子句可以使用索引。
6. 为了提高索引的效率，若 char 或 varchar 列的字符数很多，则可视具体情况选取字段的前 N 个字符值进行检索，即对索引列的前缀建立索引，这样可以节约存储空间。

第 9 章 视图

视图是数据库系统中一种非常常用的数据库对象，它往往是和外模式联系在一起的概念。外模式是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据时逻辑结构和特征的描述，通常通过数据库用户的数据视图来实现，用于对与某一应用有关的数据进行逻辑表示。

9.1 视图概述

视图是一个或多个表或者视图中导出的表，它也包含一系列带有名称的数据列和若干条数据行。然而，视图不同于数据库中真实的表，其区别在于：

1. 视图不是数据库中真实的表，是一张虚拟的表，其结构和数据是建立在对数据库中真实表的查询基础上的。
2. 视图的内容是由存储在数据库中进行查询操作的 SQL 语句来定义的，它的列数据与行数据均来自于定义视图的查询所引用的真实表（也称为基础表、基表或源表）或者是基于真实表的计算值，并且这些数据是在引用视图时动态生成的。
3. 视图不是以数据集的形式储存在数据库中，它所对应的数据实际上是存储在视图所引用的真实表（基础表）中。
4. 视图是用来查看存储在别处的数据的一种设施，其本身并不存储数据。

尽管视图与数据库中的表存在着本质的不同，但视图一经定义后，可以如同使用表一样，对视图进行查询以及受限的修改、删除和更新等操作。并且，使用视图具有如下一些优点：

1. 集中分散数据。当用户所需的数据分散在数据库的多个表中时，通过定义视图可以将这些数据集中在一起，以方便用户对分散数据的集中查询与处理。
2. 简化查询语句。定义视图可为用户屏蔽数据库的复杂性，使其不必详细了解数据库中复杂的表结构和表连接，因而能够简化用户对数据库的查询语句。例如，即便是底层数据库表发生来更改，也不会影响上层用户对数据库的正常使用，只需数据库编程人员重新定义视图的内容即可。这也正是外模式，以及模式和外模式之间映射的目的。实际上，视图的查询定义就是模式和外模式之间的映射关系。这样一来，依据数据外模式编写的应用程序就不需要更改，从而保证了数据和程序的逻辑独立性。
3. 重用 SQL 语句。视图提供的是一种对查询操作的封装，它本身不包含数据，其所呈现的数据是根据视图的定义从基础表中检索出来的，若基础表中的数据被新增或更改，视图所呈现的就是更新后的数据。因此，通过定义视图，编写完所需要的查询后，可以方便地重用该视图，而不必了解它的具体查询细节。
4. 保护数据安全。通过只授予用户使用视图的权限，而不具体指定使用表的权限，来保护基础数据的安全性。因为，通过视图用户只能查询和修改他们所能看到的数据，数

据库中的其他数据则是既看不见也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上,但不能授权到数据库的特定行上。通过视图,用户可以被限制在数据的不同子集上:使用权可以被限制在基表的行的子集上。

5. 共享所需的数据。通过视图,每个用户不必都定义和存储自己所需的数据,可以共享数据库中的数据,从而同样的数据只需存储一次。
6. 更改数据格式。通过视图,可以重新格式化检索出的数据,并组织输出到其他应用中去。

9.2 创建视图

其语法格式为:

```
create [or replace]
  view view_name [(col_list)]
  as select_statement
  [with [cascaded | local] check option]
```

主要语法说明如下:

1. or replace:可选项。用于指定 or replace 子句。该语句用于替换数据库中已有的同名视图,但需要在该视图上具有 drop 权限。
2. view_name:指定视图的名称。该名称在数据库中必须是唯一的,不能与其他表或视图同名。
3. col_list: 这个可选子句可以为视图中的每个列指定明确的名称。其中, 列名的数目必须等于 select 语句检索出的结果数据集的列数,并且每个列名之间使用逗号分隔。若省略了 col_list 子句,则新建视图使用与基础表或源视图中相同的列名。
4. select_statement:用于指定创建视图的 select 语句。这个 select 语句给出了视图的定义,它可以用于查询多个基础表或源视图。对于 select 语句的指定,存在以下一些限制:
    - 定义视图的用户除了要求被授予 create view 的权限,还必须授予可以操作视图所设计的基础表或其他视图的相关权限, 例如, 由 select 语句选择的每一列上的某些权限。
    - select 语句不能包含 from 子句中的子查询(我电脑上的 MySQL 支持 from 子句)。
    - select 语句不能引用系统变量或用户变量。
    - select 语句不能引用预处理语句参数。
    - 在 select 语句中引用的表或视图必须存在。但是,创建完视图之后,可以删除视图定义中所引用的基础表或源视图。若想检查视图定义是否存在这类问题,可以使用 check table 语句。
    - 若 select 语句中所引用的不是当前数据库的基础表或源视图时,需要在该表或视图前加上数据库的名称作为限定前缀。

- 在由 select 语句构造的视图定义中，允许使用 order by 子句。但是，如果从特定视图中进行了选择，而该视图使用了自己的 order by 语句，则视图定义中的 order by 子句将被忽略。
- 对于 select 语句中的其他选项或子句，若所创建的视图中包含了这些选项，则语句执行效果未定义。例如，如果在视图定义中包含了 limit 子句，而 select 语句也使用了自己的 limit 子句，那么 MySQL 对于使用哪一个 limit 语句未作定义。

5. with check option：这个可选子句用于指定在可更新视图上所进行的修改都需要符合 select_statement 中所指定的限定条件，这样可以确保数据修改后仍可以通过视图看到修改后的数据。但视图是根据另一个视图定义时，with check option 给出两个参数，即 cascaded 和 local，它们决定检查测试的范围。其中，关键字 cascaded 为默认值，它会对所有视图进行检查，而关键字 local 则表示只对定义的视图进行检查。

例如，在数据库 db_school 中创建视图 v_student，要求该视图包含学生信息表 tb_student 中所有男生的信息，并且保证今后对该视图数据的修改都必须符合学生性别为男性的这个条件：

```
use db_school;
create or replace view db_school.v_student
  as
  select * from db_school.tb_student where sex = '男'
  with check option;
```

例如，在数据库 db_school 中创建视图 db_school.v_score_avg，要求该视图包含表 tb_score 中所有学生的的学号和平均成绩，并按学号 studentNo 进行排序：

```
insert into db_school.tb_score
    values
      ('2013110101', '11003', 90),
      ('2013110101', '21001', 86),
      ('2013110103', '11003', 89),
      ('2013110103', '21001', 88),
      ('2013110201', '11003', 78),
      ('2013110201', '21001', 92),
      ('2013110202', '11003', 82),
      ('2013110202', '21001', 85),
      ('2013310101', '21004', 83),
      ('2013310101', '31002', 68),
      ('2013310103', '21004', 80),
      ('2013310103', '31002', 76),
      ('2014210101', '21002', 93),
      ('2014210101', '21004', 89),
      ('2014210102', '21002', 95),
      ('2014210102', '21004', 88),
      ('2014310101', '21001', 79),
      ('2014310101', '21004', 80),
```

```
    ('2014310102', '21001', 91),
    ('2014310102', '21004', 87);
create or replace view db_school.v_score_avg(学号, 平均分)
  as
  select studentNo, avg(score) from tb_score
  group by studentNo;
```

我的 MySQL 版本支持 from 子句:

```
create or replace view v_score_avgs(学号, 平均分)
  as
  select distinct t1.studentNo, t2.平均分 from tb_score as t1,
    (select studentNo, avg(score) as 平均分 from tb_score
      group by studentNo) as t2
  where t1.studentNo = t2.studentNo;
```

下面的例子反映了 with check option 的内容:

```
create view v_score
  as
  select * from db_school.tb_score where score < 50
  with check option;

create view v_score_local
  as
  select * from db_school.v_score where score > 80
  with local check option;

create view v_score_cascaded
  as
  select * from db_school.v_score where score > 80
  with cascaded check option;
```

后面两个视图是根据第一个视图定义的。local 检查只会针对其自身检查插入项进行测试;cascaded 检查则不仅会针对它自己的检查对插入项进行测试;也会针对基本视图进行测试。通过下面的语句可以清楚得分辨彼此之间的差异:

```
insert into db_school.v_score_local
    values
      ('2013110101', '21005', 90);

--结果

[2018-09-01 15:04:05] [HY000][1369] CHECK OPTION failed 'db_school.
    v_score_local'
```

9.3 删除视图

```sql
drop view if exists db_school.v_student;
```

9.4 修改视图定义

例如：

```sql
-- 因为刚刚删除了，所以再重新定义一个
create or replace view db_school.v_student
  as
  select * from db_school.tb_student where sex = '男'
  with check option;
-- 修改
alter view db_school.v_student(studentNo, studentName, classNo)
as
select studentNo, studentName, classNo from db_school.tb_student
where sex = '男' and nation = '汉'
with check option;
```

或者使用 create or replace 语句也可以直接修改。

```sql
create or replace view db_school.v_student
  as
  select studentNo, studentName, classNo from db_school.tb_student
  where sex = '男' and nation = '汉'
  with check option;
```

9.5 查看视图定义

```sql
show create view db_school.v_student;
```

或者在终端：

```
mysql> show create view db_school.v_student \G;
*************************** 1. row ***************************
                View: v_student
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
             SQL SECURITY DEFINER VIEW `db_school`.`v_student` AS select `
             db_school`.`tb_student`.`studentNo` AS `studentNo`,`db_school`.`
             tb_student`.`studentName` AS `studentName`,`db_school`.`
             tb_student`.`classNo` AS `classNo` from `db_school`.`tb_student`
              where ((`db_school`.`tb_student`.`sex` = '男') and (`db_school
             `.`tb_student`.`nation` = '汉')) WITH CASCADED CHECK OPTION
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
1 row in set (0.01 sec)
```

9.6 更新视图数据

由于视图是个虚拟表，所以通过插入、修改和删除等操作方式来更新视图中的数据，实质上是在更新视图所引用的基础表中的数据。然而，视图的更新操作是受一定限制的，并非所有的视图都可以进行 insert、update 或 delete 等更新操作，只有满足可更新条件的视图才能进行更新，否则可能会导致系统出现不可预期的结果。对于可更新的视图，需要该视图中的行和基础表中的行具有一对一的关系。另外，倘若视图中包含了下述任何一种 SQL 语句结构，那么该视图就是不可更新的：

1. 聚合函数；
2. distinct 关键字；
3. group by 子句；
4. order by 子句；
5. having 子句；
6. union 运算符；
7. 位于选择列表中的子查询；
8. from 子句中包含多个表；
9. select 语句中引用了不可更新的视图；
10. where 子句中的子查询引用了 from 子句中的表。

9.6.1 使用 insert 语句通过视图向基础表中插入数据

```
create or replace view db_school.v_student
  as
  select * from db_school.tb_student
  with check option ;

insert into db_school.v_student
    values
      ('2014310108', '周明', '男', '1997-08-16', '辽宁', '汉', 'IS1401');

select *
from db_school.v_student;
```

with check option 子句会在更新数据时检查新数据是否符合视图定义中 where 子句的条件，并且 with check option 子句只能和可更新视图一起使用。若插入的新数据不符合 where 子句的条件，则数据插入操作不会成功，因而此时视图的数据插入操作受限。另外，但视图所依赖的的基础表有多个时，也不能向该视图插入数据，这是因为 MySQL 不能正确地确定要被更新的基础表。

9.6.2 使用 update 语句通过视图修改基础表的数据

```
# 将视图v_student中所有学生的native都更新为"河南"
```

```
update db_school.v_student
    set native = '河南';
```

9.6.3 使用 delete 语句通过视图删除基础表的数据

```
delete from db_school.v_student
where studentName = '周明';
```

9.7 查询视图数据

视图一经定义后，就可以如同查询数据库中的表一样，对数据库进行数据查询，这也是对视图使用最多的一种操作。视图用于查询检索，主要体现在这样一些应用：

1. 利用视图简化复杂的表连接；
2. 使用视图重新格式化检索出的数据；
3. 使用视图过滤不想要的数据。

例如，在视图 v_student 中查找 classNo 为 “CS1401'' 的学生学号和姓名：

```
select studentNo, studentName
from db_school.v_student
where classNo = 'CS1401';
```

9.8 对视图的进一步说明

在 MySQL 中，对视图的使用还需要注意以下几点：

1. 创建视图必须有足够的权限；
2. 对于可以创建的视图数目没有限制；
3. 视图可以嵌套，既可以利用从其他视图中检索数据的查询来构造一个视图。
4. order by 子句可以用在视图中。但如果从该视图检索数据的 select 语句也包含 order by 子句，那么该视图中的 order by 子句将被覆盖；
5. 视图不能被索引，也不能有关联的触发器、默认值。
6. 视图可以和表一起使用。例如，编写一条连接表和视图的 select 语句；
7. 由于视图不包含数据，所以每次使用视图时都必须处理查询执行是所需的任何一个检索操作。倘若用多个连接和过滤条件创建来复杂的视图或者嵌套来视图，可能会发现系统运行性能下降得非常严重。因此，在部署使用了大量视图的应用前，应该进行性能测试。

第 10 章 触发器

触发器是 MySQL 5.0 开始支持的一种过程式数据库对象。

10.1 触发器

触发器是一个被指定关联到一个表的数据库对象，但对一个表的特定事件出现时，它将会被激活。触发器具有 MySQL 语句在需要时才被执行的特点，即某条（或某些）MySQL 语句在特定事件发生时自动执行。例如：

1. 每当增加一个客户到数据库的客户基本信息表时，都检查其电话号码的格式是否正确；
2. 每当客户订购一个产品时，都从产品库存中减去订购的数量。
3. 每当删除客户基本信息表中一个客户的全部信息数据时，该客户所订购的未完成订单信息也应该自动被删除。
4. 无论何时删除一行，都在数据库的存档中保留一个副本。

触发器与表的关系非常密切，用于保护表中的数据。当有操作影响到触发器所保护的数据时，触发器就会被自动执行，从而保障数据库中数据的完整性，以及多个表之间数据的一致性。具体而言，触发器就是 MySQL 响应 insert、update 和 delete 语句自动执行的一条 MySQL 语句（或位于 begin 和 end 语句之间的一组 MySQL 语句）。需要注意的是，其他 MySQL 语句是不支持触发器的。

10.2 创建触发器

在 MySQL5.5 中，可以使用 create trigger 语句创建触发器，其语法格式为：

```
create
  trigger trigger_name trigger_time trigger_event
  on tbl_name for each row trigger_body
```

语法说明如下：

1. trigger_name：触发器的名称，触发器在当前数据库中必须具有唯一的名称。如果要在某个特定的数据库中创建，名称前面应该加上数据库的名称。
2. trigger_time：触发器被触发的时刻，它有两个选项，即 before 和 after，用于表示触发器是在激活它的语句之前或者之后激发。如果希望验证新数据是否满足使用的限制，则使用 before；如果希望在激活触发器的语句执行之后完成几个或更多的改变，通常使用 after 选项。
3. trigger_event：触发事件，用于指定激活触发器的语句的种类，其可以是下述值之一：

- insert：将新行插入表时激活触发器。例如，insert 的 before 触发器不仅能被 MySQL 的 insert 语句激活，也能被 load data 语句激活。
- update：更改表中的某一行时激活触发器。
- delete：删除某一行时激活触发器。

4. tbl_name：与触发器关联的表名，必须是永久性表，不能将触发器与临时表或视图关联起来。在该表上触发事件发生时才会激活触发器。同一个表不能拥有两个具有相同触发时刻和事件的触发器。例如，对于一张数据表，不能同时拥有两个 before update 触发器，但是可以有一个 before update 触发器和一个 before insert 触发器，或一个 before update 触发器与一个 alter update 触发器。
5. for each row：这个声明用来指定对于受触发事件影响的每一行都要激活触发器的动作。例如，使用一条 insert 语句向一个表中插入多行数据时，触发器会对每一行数据的插入都执行相应的触发操作。
6. trigger_body：触发器动作主体，包含触发器激活时将要执行的 MySQL 语句。如果要执行多个语句，可使用 begin. . . end 复合语句结构。这里可使用存储过程中允许的相同语句。

注意，在触发器的创建中，每个表每个事件每次只允许一个触发器。因此，每个表最多支持六个触发器，即每条 insert、update 和 delete 的前后。单一触发器不能与多个事件或多个表关联，例如，需要一个对 insert 和 update 操作执行的触发器，则应该定义两个触发器。

另外，在 MySQL 中，如果需要查看数据库中已有的触发器，可以使用 show triggers 语句，其语法格式为：

```
show triggers [{from | in} db_names];
```

例如，在数据库 db_school 的表 tb_student 中创建一个触发器 tb_student_insert_trigger，用于每次向表 tb_student 中插入一行数据时将学生变量 str 的值设置为 “one student added! ’’：

```
create trigger db_school.tb_student_insert_trigger
  after insert
  on db_school.tb_student
  for each row
  set @str = '一个学生被添加! ';

# 添加一个学生
insert into db_school.tb_student
    values
      ('2013120202', '张大勇', '男', '1997-12-11', '山西', '汉', 'AC1301');
# 验证触发器
select @str;
```

10.3 删除触发器

drop trigger 需要 super 权限。当删除一个表的同时也会自动地删除这个表上的触发器。另外触发器不能更新或覆盖,为了修改一个触发器,必须先删除再重新创建。

```
drop trigger if exists db_school.tb_student_insert_trigger;
```

10.4 使用触发器

实际使用中,MySQL 所支持的触发器有三种:insert 触发器、delete 触发器和 update 触发器。

10.4.1 insert 触发器

使用该触发器时需要注意以下几点:

- 在 insert 触发器代码内可以引用一个名为 new 的虚拟表来访问插入的行;
- 在 before insert 触发器中,new 中的值可以被更新,即允许更改被插入的值(只要具有对应的操作权限)。
- 对于 auto_increment 列,new 在 insert 执行之前包含的是 0 值,在 insert 执行之后将包含新的自动生成值。

例如,重新创建 tb_student_insert_trigger,用于每次向表 tb_student 中插入一条数据时将学生变了 str 的值设置为新插入学生的学号:

```
create trigger db_school.tb_student_insert_trigger
  after insert
  on db_school.tb_student
  for each row
  set @str = new.studentNo;

# 然后插入一行数据
insert into db_school.tb_student
    values
      ('2013330101', '张三勇', '男', '1997-12-11', '山西', '汉', 'AC1301');

# 验证触发器
select @str;
```

10.4.2 delete 触发器

delete 触发器代码内可以引用一个名为 old 的虚拟表来访问被删除的行。old 中的值全部是只读的,不能被更新。

10.4.3 update 触发器

update 触发器既可以引用一个名为 old 的虚拟表访问之前的值，还可以引用一个名为 new 的虚拟表访问新的值。

在 before update 触发器中，new 中的值可能被更新，即允许更改将要用于 upadate 语句中的值。

当触发器涉及对表自身的更新操作时，只能使用 before update 触发器，而 alter update 触发器将不被允许。

例如，在 db_school.tb_student 中创建以恶搞 tb_student_update_trigger，用于每次更新表时将该表中 nation 列的值设置为 native 列的值。

```
create trigger db_school.tb_student_update_trigger
  before update
  on db_school.tb_student
  for each row
  set new.nation = old.native;

# 更新一个数据
update db_school.tb_student set nation = '壮'
where studentName = '张三勇';
# 执行下面的语句会发现，nation列的值为"山西"而非壮
select nation from db_school.tb_student
where studentName = '张三勇';
```

在触发器的执行过程中，MySQL 会安装下面的方式来处理错误：

1. 如果 before 触发程序失败，则 MySQL 将不执行相应行上的操作；
2. 仅当 before 触发程序和行操作都被成功执行，MySQL 才会执行 after 触发程序（如果有的话）；
3. 如果在 before 或 after 触发程序的执行过程中出现错误，将导致调用触发程序的整个语句的失败。

10.5 对触发器的进一步说明

1. 与其他 DBMS 相比，目前 MySQL 版本支持的触发器还较为低级；
2. 创建触发器可能需要特殊的安全访问权限，但是触发器的执行是自动的。
3. 应该多用触发器来保证数据的一致性、完整性和正确性。例如，使用 before 触发程序进行数据的验证和净化，这样保证插入表的数据是所需要的正确数据，而且这种操作对用户是透明的。触发器是一种非常有意义的使用模式——--创建审计跟踪，也就是可以使用触发器把表的更改状态以及之前和之后的状态记录到另一张数据表中。

第 11 章 事件

事件是一类过程式数据库对象。本文重点介绍了创建、修改和删除事件的 SQL 语法。

11.1 事件

事件可以通过 MySQL 服务器中一个非常有特色的功能模块——--事件调度器进行监视，并判断其是否需要被调用。事件调度器可以在指定的时刻执行某些特定的任务，并以此可取代原先只能由操作系统的计划任务来执行的工作。这种需要在指定时刻才被执行的某些特定任务就是事件，这些特定任务通常是一些确定的 SQL 语句。目前，MySQL 可的事件调度器可以精确到每秒钟执行一个任务，这对于一些对于数据实时性要求较高的应用非常适合，例如股票、赔率和比分等。

事件和触发器类似，都是在某些事情发生的时候启动，因此事件也可称为临时触发器。其中，事件是基于特定时间周期触发来执行某些任务，而触发器是基于某个表所产生的事件触发的，它们之间的区别也在于此。

在使用事件调度器这个功能之前，必须确保 MySQL 中 event_scheduler 已被开启。这个可以通过如下语句检查：

```
show variables like 'event_scheduler';
# 或查看系统变量：
select @@event_scheduler;
```

如果没有开启，可以通过下面的语句开启：

```
# 开启event_scheduler
set global event_scheduler = true;
# 或
set global event_scheduler = 1;
```

再或者，可以再 MySQL 的配置文件 my.ini 中加上：

`event_scheduler=1` 或者 `set global event_scheduler=on`

11.2 创建事件

语法格式为：

```
create event [if not exists] event_name
  on schedule schedule
  [enable | disable | disable on slave]
  do event_body
```

其中，schedule 的语法格式为：

```
at timestamp [+ interval interval] ...
| every interval
[starts timestamp [+ interval interval] ... ]
[ends timestamp [+ interval] ... ]
```

interval 的语法格式为:

```
quantity {year | quarter | month | day | hour | minute |
week | second | year_month | day_hour | day_minute |
day_second | hour_minute | hour_second | minute_second}
```

主要语法说明如下:

1. event_name:指定事件名,前面可以添加 if not exists 来修饰;
2. schedule:时间调度,用于指定事件何时发生或者每隔多久发生一次,分别对应下面两个子句:
   - at 子句:用于指定事件在某个时刻发生。其中,timestamp 表示一个具体的时间点,后面可以加上一个时间间隔,表示在这个时间间隔后事件发生;interval 表示这个事件间隔由一个数值和单位构成;quantity 是间隔时间的数值。
   - every 子句:用于表示事件在指定事件区间内每间隔多长事件发生一次。其中 starts 子句用于指定开始事件,ends 子句用于指定结束事件。
3. event_body:do 子句中的 event_body 部分用于指定事件启动时所需要执行的代码。如果包含多条语句,可以使用 begin...end 复合结构。
4. enable | disable | disable on slave:为可选项,表示事件的一种属性。其中,关键字 enable 表示该事件是活动的,意味着调度器检查事件动作是否必须调用。关键字 disable 表示该事件是关闭的,关闭意味着事件的什么存储到目录中,但是调度器不会检查它是否应该调用;关键字 disable on slave 表示事件在从机中是关闭的。如果不指定三个选项中的任何一个,这则在一个事件创建之后,它立即变为活动的。

例如,在数据库 db_school 中创建一个事件,用于每个月向表中插入一条数据,该事件开始于下个月并且在 2018 年 12 月 31 日结束:

```
drop event if exists event_insert;
create event if not exists event_insert
  on schedule every 1 month
  starts curdate() + interval 1 month
  ends '2018-12-31'
  do
  begin
    if year(curdate()) < 2019 then
      insert into tb_student
        values
          (null, '张四勇', '男', '1997-12-11', '山西', '汉', 'AC1301');
    end if;
  end;
```

11.3 修改事件

其语法格式为:

```
alter event event_name
[on schedule schedule]
[rename to new_event_name]
[enable | disable | disable on slave]
[do event_body]
```

alter table 语句与 create event 语句使用的语法相似,这里不再重复解释。用户可以使用一条 alter event 语句让一个事件关闭或再次让其活动。需要注意的是,一个事件最后一次被调用后,是无法被修改的,因为此时它已经不再存在了。例如,临时关闭上面创建的事件:

```
# 临时关闭上面创建的事件
alter event event_insert disable;

# 再次开启
alter event event_insert enable;

# 改个名字
alter event event_insert
rename to  e_insert;
```

11.4 删除事件

```
drop event if exists e_insert;
```

第 12 章 存储过程与存储函数

存储过程和存储函数是 MySQL 自 5.0 版本之后开始支持的过程式数据库对象。他们作为数据库存储的重要功能，可以有效提高数据库的处理速度，同时也可以提高数据库编程的灵活性。

12.1 存储过程

存储过程是一组为了完成某特定功能的 SQL 语句集，其实质上就是一段存放在数据库中的代码，它可由声明式的 SQL 语句（如 create、update 和 select 等语句）和过程式 SQL 语句（如 if-then-else 控制结构语句）组成。这组语句集经过编译后会存储在数据库中，用户只需要通过指定存储过程的名字并给定参数（如果该存储过程带有参数），即可随时调用并执行它，而不必重新编译，因此这种通过定义一段程序存放在数据库中的方式，可加大数据库操作语句的执行效率。而前面介绍的各条 MySQL 数据库操作语句在其执行过程中，需要首先编译，然后执行。尽管这个过程由 DBMS 自动完成，而且对 SQL 语句的使用者透明，但这种每次执行之前都需要预先编译，就成了数据库操作语句执行效率的一个瓶颈问题。

一个存储过程是一个可编程的函数，同时可看作是在数据库编程中面向对象方法的模拟。它允许控制数据的访问方式，因为，当希望在不同的应用程序或平台上执行相同的特定功能时，存储过程尤为合适，其通常具有这样一些优点：

1. 可增强 SQL 语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。
2. 良好的封装性。存储过程被创建之后，可以在程序中被多次调用，而不必重新编写该存储过程的 SQL 语句，并且数据库专业人员可以随时对存储过程进行修改，而不会影响到调用它的应用程序源代码。
3. 高性能。存储过程执行一次后，其执行规划就驻留在高速缓冲存储器中，在以后的操作中，只需从高速缓冲存储器中调用已编译好的二进制代码执行即可，从而提高了系统性能。
4. 可减少网络流量。由于存储过程是在服务器端运行，且执行速度快，那么当客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而可降低网络负载。
5. 存储过程可作为一种安全机制来确保数据库的安全性和数据的完整性。使用存储过程可以完成所有数据库操作，并可以通过编程方式控制这些数据库对数据库信息访问的权限。

12.1.1 创建存储过程

其语法格式如下：

```
create procedure sp_name([proc_parameter[, ... ]])
  [characteristic ... ] routine_body
```

其中,proc_parameter 的格式为:

```
[in | out | inout] param_name type
```

type 的格式为:

```
any valid mysql data type
```

characteristic 的格式为:

```
comment 'string'
| language sql
| [not] deterministic
| {contains sql | no sql | reads  sql data | modifies sql data}
| sql security {definer | invoker}
```

routine_body 的格式为:

```
valid sql routine statement
```

主要语法说明如下:

1. sp_name:存储过程的名称,默认在当前数据库中创建。需要注意避免与 MySQL 的内置函数重名。
2. proc_parameter:存储过程的参数列表。其中,param_name 为参数名,type 为参数的类型(可以为任何有效的 MySQL 数据类型)。当有多个参数时,参数列表中彼此之间用逗号隔开。存储过程可以没有参数,此时存储过程的名称的后面也需要加一对括号,也可以有一个或多个参数。MySQL 存储过程支持三种类型的参数,即输入参数、输出参数和输入输出参数,分别用 in、out 和 inout 三个关键字标识。其中,输入参数是使数据可以传递给一个存储过程;输出参数用于存储过程需要返回一个操作结果的情形;而输入输出参数既可以充当输入参数也可以充当输出参数。需要注意的是,参数的取名不要与数据表的列名相同,否则尽管不会返回错误信息,但是存储过程中的 SQL 语句会把这些参数名看成列名,从而引发不可预知的结果。
3. characteristic:存储过程的某些特征设定,下面分别介绍:
    - comment ‘string’:用于对存储过程的描述,其中 string 为描述内容,comment 为关键字,这个描述信息可以使用 show create procedure 语句来显示。
    - language sql:指名编写这个存储过程的语言是 SQL 语言。目前位置 MySQL 存储过程还不能用外部编程语言来编写,也就是说,这个选项可以不指定。今后 MySQL 将会对其进行扩展,最有可能第一个被支持的语言是 PHP。
    - deterministic:若设置为 deterministic,表示存储过程对同样的输入参数产生相同的结果;若设置为 not deterministic,则会产生不确定的结果。默认为 not deterministic。

- contains sql | no sql | reads sql data | modifies sql data：contains sql 表示存储过程包含读或写数据的语句；no sql 表示存储过程不包含 sql 语句；reads sql data 表示存储过程包含读数据的语句，但是不包含写数据的语句；modifies sql data 表示存储过程包含写数据的语句。若没有明确给定，则默认 contains sql。
- sql security：这个特征用来指定存储过程使用创建该存储过程的用户 (definer) 的许可来执行，还是使用调用者 (invoker) 的许可来执行。没有明确指定时，默认为 definer。

4. routine_body：存储过程的主体部分，也称为存储过程体，其包含了在过程调用的时候必须执行的 SQL 语句。这个部分是以关键字 begin 开始，并以关键字 end 结束。若存储过程中只有一条 SQL 语句时，可以省略 begin-end 标识。另外，在存储过程体中，begin-end 复合语句还可以嵌套使用。

在存储过程的创建过程中，经常会用到一个非常终于的 MySQL 命令，即 delimiter 命令。特别是对于通过命令行的方式操作 MySQL 数据库的使用者，更是要学会使用该命令。

在 MySQL 中，服务器处理 SQL 语句默认是以分号作为语句结束的标志。然而，在创建存储过程时，存储过程体中可能包含多条 SQL 语句，这些 SQL 语句如果仍以分号作为语句结束符，那么 MySQL 服务器在处理时会以遇到的第一条 SQL 语句结尾处的分号作为整个程序的结束符，而不再去处理存储过程体后面的 SQL 语句，这样显然是不行的。为解决这个问题，通常可以使用 delimeter 命令，将 MySQL 语句的结束标志临时修改为其他符号，从而使得 MySQL 服务器可以完整地处理存储过程体中所有的 SQL 语句。

delimeter 命令的语法格式是：

```
delimeter $$
```

语法说明如下：

1. $$ 是用户定义的结束符号。通常这个符号可以是一些特殊的符号，例如 ## 或 ěě 等。
2. 当使用 delimeter 命令时，应该避免使用反斜杠，因为它是 MySQL 的转义字符。

例如，将 MySQL 结束符号修改为两个感叹号：

```
# 将MySQL结束符号修改为两个感叹号（这个需要在终端运行）
delimiter !!
```

在 MySQL 中创建一个存储过程，用于实现给定表 tb_student 中一个学生的学号即可修改表 tb_student 中该学生的性别为另一个指定性别。

```
use db_school;
delimiter $$
create procedure sp_update_sex(in sno char(20), in ssex char(2))
  begin
    update tb_student set sex = ssex where studentNo = sno;
  end $$
```

使用说明，在 MySQL5.5 中创建存储过程，必须具有 create routine 权限。若要查看数据库中有哪些存储过程，可以使用 show procedure status 命令；若要查看某个存储过程的

具体信息,则可以使用 show create procedure sp_name 命令,其中 sp_name 用于指定该存储过程的名称。

12.1.2 存储过程体

在存储过程中可以使用各种 SQL 语句与过程式语句的组合,来封装数据库应用中复杂的业务逻辑和处理规则,以实现数据库应用的灵活编程。这里,主要介绍几个用于构造存储过程体的常用语法元素。

12.1.2.1 局部变量

在存储过程体中可以声明局部变量,用来存储存储过程体的临时结果。在 MySQL5.5 中,可以使用 declare 语句来声明局部变量,并且同时还可以对该局部变量赋予一个初始值。其语法格式是:

```
declare var_name [, ...] type [default value]
```

语法说明如下:

1. var_name:用于指定局部变量的名称;
2. type:用于声明局部变量的数据类型;
3. default 子句:用于为局部变量指定一个默认值。若没有指定,则默认为 null;

例如,声明一个整型局部变量 sno。在存储过程中可以使用如下语句来实现:

```
declare sno char(10);
```

使用说明:

- 局部变量只能在存储过程体的 begin...end 语句块中声明;
- 局部变量必须在存储过程体的开头处声明;
- 局部变量的作用范围仅限于声明它的语句块中;
- 局部变量不同于用户变量,两者的区别是:@,并且它只能在声明它的语句块中被使用;而用户变量存在整个会话之中。

12.1.2.2 set 语句

在 MySQL5.5 中,可以使用 set 语句为局部变量赋值,其语法格式是:

```
set var_name = expr[, var_name = expr]...
```

例如:

```
set sno = '2013110101'
```

12.1.2.3 select ... into 语句

在 MySQL5.5 中,可以使用 select...into 语句把选定列的值直接存储到局部变量中,其语法格式是:

```
select col_name [, ... ] into var_name [, ... ] table_expr
```

- table_expr:表示 select 语句中的 from 子句及后面的语法部分。
- 存储过程体的 select. . . into 语句返回的结果集只能有一行数据。

12.1.2.4 流程控制语句

在 MySQL5.5 中，可以在存储过程体中使用以下两类用于控制语句流程的过程式 SQL 语句:

12.1.2.4.0.1 条件判断语句 常用的条件判判断语句有 `if-then-else` 语句和 `case` 语句。其中 `if-then-else` 语句的语法格式如下:

```
if search_condition then statement_list
  [elseif search_condition then statement_list] ...
  [else statement_list]
end if
```

- `if-then-else` 语句不同于系统内置函数 if().

case 语句在存储过程中使用具有两种语法格式,分别是:

```
case case_value
  when when_value then statement_list
  [when when_value then statement_list] ...
  [else statement_list]
end case
```

或者

```
case
  when search_condition then statement_list
  [when search_condition then statement_list] ...
  [else statement_list]
end case
```

12.1.2.4.0.2 循环语句

1. while 语句:

```
[begin_label:]
  while search_condition
    do statement_list
  end while
[end_label]
```

1. repeat 语句

```
[begin_label:] repeat
  statement_list
  until search_condition
end repeat
[end_label]
```

1. loop 语句

```
[begin_label:] loop
    statement_list
end loop
[end_label]
```

此时，在循环体中的语句会被一直重复执行，直至循环使用 leave 语句退出，leave 语句的语法格式为:`leave label`,这里的 label 是 loop 语句中所标注的自定义名字。

另外,循环语句中还可以使用 iterate 语句,但它只能出现在循环语句的 loop、repeat 和 while 子句中,用于表示退出当前循环,且重新开始一个循环。其语法格式为:iterate label,这里的 label 同样是循环语句中自定义的标注名字,iterate 语句与 leave 语句的区别在于,leave 语句是结束整个循环,而 iterate 语句只是退出当前循环,然后开始下一轮循环。

12.1.2.5 游标

在 MySQL 中,一条 select. . . into 语句成功执行后,会返回带有值的一行数据,这行数据可以被读取到存储过程中进行处理。然后使用 select 语句进行数据检索时,若该语句被成功执行,则会返回一组被称为结果集的数据行,该数据集中可能拥有多行数据,这些数据无法直接被一行一行的处理,此时就需要使用游标。游标是一个被 select 语句检索出来的结果集。在存储了游标之后,应用程序或用户就可以根据需要滚动或浏览其中的数据。

在目前版本的 MySQL 中,若要使用游标,需要注意以下几点:

1. MySQL 对游标的支持是从 MySQL 5.0 开始的,之前的版本不支持游标;
2. 游标只能用于存储过程或存储函数中,不能单独在查询操作中使用。
3. 在存储过程或存储函数中可以定义多个游标,但是在一个 begin. . . end 语句块中每一个游标的名字必须是唯一的。
4. 游标不是一条 select 语句,而是被 select 语句检索出来的结果集。

在 MySQL 5.5 中,使用游标的具体步骤如下:

1. 声明游标

在使用游标之前，必须先声明它。这个过程实际上没有检索数据，只是定义使用的 select 语句。在 MySQL5.5 中,可以使用 declare cursor 语句来创建游标,语法格式如下:

```
declare cursor_name cursor for select_statement;
```

语法说明如下:

- cursor_name:指定要创建的游标的名称,其命名规则与表名相同;

- sleect_statement：指定一个 select 语句，其会返回一行或多行数据。注意：这里的 select 语句中不能有 into 子句；

1. 打开游标

在定义游标之后，必须打开游标才能使用。这个过程实际上是将游标连接到 select 语句返回的结果集中。在 MySQL5.5 中，可以使用 open 语句打开游标，语法格式如下：

```
open cursor_name;
```

在实际应用中，一个游标可以被多次打开，由于其他用户或应用程序可能随时更新来数据表，因此每次打开游标的结果集可能会不同。

1. 读取数据

对于填有数据的游标，可以根据需要取出数据。在 MySQL5.5 中，可以使用 fetch. . . into 语句从从读取数据，语法格式为：

```
fetch cursor_name into var_name [, varname]...;
```

fetch. . . into 语句与 slect. . . into 语句具有相同的意义，fetch 语句是将游标指向的一行数据赋给一些变量，这些变量的数据必须声明游标时 select 子句中选择列的数目。游标相当于一个指针，它指向当前的一行数据。

2. 关闭游标

在结束游标使用时，必须关闭游标。语法格式为：

```
close cursor_name;
```

每个游标不再需要使用的时候都应该被关闭，使用 close 语句将会释放游标所使用的全部资源。在一个游标被关闭后，如果没有重新被打开，则不能被使用。对于声明过的游标，则不需要再次声明，只需要直接使用 open 打开即可。另外，如果没有明确关闭游标，MySQL 会在到达 end 语句时自动关闭它。

例如，在数据库 db_school 中创建一个存储过程，用于计算表 tb_student 中数据行的行数。

```
use db_school;
delimiter $$
drop procedure if exists sp_sumofrow;
create procedure sp_sumofrow (out myrows int)
  begin
    declare sno char(10);
    declare found boolean default true;
    declare cur cursor for select studentNo from db_school.tb_student;
    declare continue handler for not found set found = false;
    set myrows = 0;
    open cur;
    fetch cur into sno;
    while found do
      set myrows = myrows + 1;
      fetch cur into sno;
```

```
    end while ;
    close cur;
  end $$

call sp_sumofrow(@myrows);

select  @myrows;
```

这里我遇到了非常难以解决的错误。最后我幸运的发现,把书上的 rows 换成 myrows 并且为 sno 的类型声明为 char(10) 就不会出错,猜想这大概是因为 rows 不能作为参数名吧。

上例中定义例一个 continue handler 句柄,它是在条件出现时被执行的代码,用于控制循环语句,以实现游标的下移。declare 语句的使用存在特定的次序。用 declare 语句定义的局部变量必须在定义任意游标会句柄之前定义,而句柄必须在游标之后定义。否则系统会出现错误信息。

12.1.3 调用存储过程

语法格式为:

```
call sp_name([parameter[, ... ]]);
call sp_name[()];
```

语法说明如下:

- sp_name:指定被调用的存储过程的名称。如果调用某个特定数据库的存储过程,则需要在前面加上该数据库的名称;
- parameter:指定调用存储过程所要使用的参数。调用语句中参数的个数必须等于存储过程的参数个数;
- 当调用没有参数的存储过程时,使用 call sp_name() 语句与使用 call sp_name 语句是相同的。

例如,调用数据库 db_school 中的存储过程 sp_update_sex,将学号为 "2013110201'' 的学生性别修改为 ''男":

```
call sp_update_sex('2013110201', '男');
select * from tb_student where studentNo = '2013110201';
```

12.1.4 删除存储过程

上面的代码中已经展示过。

12.2 存储函数

在 MySQL 中,存在一种与存储过程非常相似的过程式数据库对象——--存储函数。它与存储过程一样,都是有 SQL 语句和过程式语句所组成的代码片段,并且可以被应用程

序和其它 SQL 语句调用。但是存储函数和存储过程仍然存在以下几点区别:

- 存储函数不能拥有输出参数。这是因为存储函数自身就是输出参数;而存储过程可以拥有输入参数和输出参数以及输入输出参数;
- 可以直接对存储函数进行调用,且不需要使用 call 语句;而对存储过程的调用需要使用 call 语句;
- 存储函数必须包含一条 return 语句,而这条特殊的 return 语句不允许包含于存储过程中。

创建存储函数

语法格式为:

```
create function sp_name ([func_parameter[, ...]])
  returns type
  routine_body
```

其中,func_parameter 的格式为:

```
param_name type
```

type 的格式为:

```
Any valid MySQL data type
```

routine_body 的格式为:

```
Valid SQL routine statement
```

由于 create function 语句与前面的 create procedure 语句的语法大致相同,故这里仅对两者有区别的地方进行说明:

- sp_name:用于指定存储函数的名称。注意,存储函数不能与存储过程同名。
- func_parameter: 用于声明存储函数的参数。这里的参数只有名称和类型,不能指定关键字 in/out/inout。
- returns 子句:用于声明存储函数返回值的数据类型。
- routine_body: 存储函数的主体部分,也称为存储函数体。所有在存储过程中使用的 SQL 语句在存储函数中同样适用,包括前面介绍的局部变量、set 语句、流程控制语句、游标等。但是,存储函数体还必须包含一个 return value 语句,其中的 value 用于指定存储函数的返回值。

例如,在数据库 db_school 中创建一个存储函数,要求该函数能够根据给定的学号返回学生的性别,如果数据库中没有给定的学号,就返回没有该学生。

```
use db_school;
delimiter $$
drop function if exists db_school.fn_search;
create function fn_search(sno char(10))
  returns char(10)
  deterministic
  begin
```

```
    declare ssex char(2);
    select sex into ssex from tb_student
      where studentNo = sno;
    if ssex is null then
      return (select '没有该学生');
    else if ssex = '女' then
        return (select '女');
      else return (select '男');
      end if ;
    end if ;
  end $$

select fn_search('2013110201');
select fn_search('2014110201');
```

第 13 章 访问控制与安全管理

数据库服务器通常包含有关键的数据，这些数据的安全和网站可通过访问控制来维护。MySQL 中提供来访问控制，以此确保 MySQL 服务器的安全访问，即用户应该对他们需要的数据具有适当的访问权，既不能多也不能少。因此，MySQL 的访问控制就是为用户提供且仅提供他们所需的访问权。

13.1 用户账号管理

MySQL 的用户账号和相关信息存储在一个名为 mysql 的 MySQL 数据库中，这个数据库里有一个名为 user 的数据表，包含来所有用户账号，并且它用一个名为 user 的列存储用户的登陆名。这里，可以使用下面的 SQL 语句查看 MySQL 数据库的使用账号。

```
select user from mysql.user;
```

可以看到 root 用户拥有对整个 MySQL 服务器完全控制的权限。在对 MySQL 的日常管理和实际操作中，为了避免恶意用户冒名使用 root 账号操纵数据库，通常需要创建一些列具备适当权限的账号，而尽可能地不用或少用 root 账号登陆系统，以此来确保数据的安全访问。

13.1.1 创建用户账号

可以使用 create user 语句来创建一个或多个 MySQL 账户，并且设置相应的口令，其语法格式为：

```
create user user_specification
  [, user_specification]...
```

其中，user_specification 的格式为：

```
user
[
  identified by [password] 'password'
  | identified with auth_plugin [as 'auth_string']
]
```

语法说明如下：

- user：指定创建的用户账号。其格式为 ‘$\texttt{user}_{\texttt{n}}\texttt{ame@host}$ name’。这里，user_name 是用户名，host_name 为主机名，即用户连接 MySQL 时所在主机的名字。如果在创建的过程中，只给出了账户中的用户名，而没有设定主机名，这主机名会被默认为是 “%’’，表示一组主机。

- identified by 子句:用于指定用户账号对应的口令,若该用户账号无口令,则可省略此子句。
- 可选项 password:用于指定散列口令,即若使用明文设置口令,需忽略 password 关键字;如果不想以明文设置口令,且知道 password() 函数返回给密码的散列值,这可以在此口令设置语句中指定此散列值,但需要加上关键字 password。
- password: 指定用户账号和口令,在 identified by 关键字或 password 关键字之后,给定的口令值可以是只由字母和数字组成的明文,也可以是通过 password() 函数得到的散列值。
- identified with 子句:用于指定验证用户账号的认证插件。
- auth_plugin:指定认证插件的名称,紧跟在 identified with 关键字之后。

注意:在 MySQL8.0 中,password() 函数已经被删除了!

例如,在 MySQL 服务器中添加两个新的用户,其用户名分别为 zhangsan 和 lisi,他们的主机名均为 localhost,密码分别为名为 “123’’ 和 “456’’。

```
# 在 MySQL 服务器中添加两个新的用户, 其用户名分别为zhangsan和lisi,
# 他们的主机名均为localhost, 密码分别为名为“123”和“456”。
select md5('456');
create user
  'zhangsan'@'localhost' identified by '123',
  'lisi'@'localhost' identified by '456';
```

create user 语句的使用说明如下:

- 要使用 create user 语句,必须拥有 MySQL 中 mysql 数据库的 insert 权限或全局 create user 权限。
- 使用 create user 语句创建一个用户账号之后,会在系统自身的 mysql 数据库的 user 表中添加一条新纪录。如果创建的账户已经存在,则语句执行会出错。
- 如果两个用户具有相同的用户名和不同的主机名,MySQL 会将他们视为不同的用户,并允许为则两个用户分配不同的权限集合。
- 如果 create user 语句的使用中没有为用户指定口令,那么 MySQL 允许用户可以不使用口令登陆账户。
- 新茶创建的用户拥有的权限很少。他们可以登录到 MySQL,只允许进行不需要权限的操作,如使用 show 语句查询所有存储引擎和字符集的列表等。不能使用 use 语句来让其他用户已经创建的任何数据库成为当前数据库,因而无法访问相关数据库的表。

13.1.2 删除用户

```
drop user user [, user] ...
```

语法说明如下:

- drop user 语句可以用于删除一个或多个 MySQL 账户,并消除其权限。

- 要使用 drop user 语句，必须拥有 MySQL 中 mysql 数据库的 delete 权限或全局 alter user 权限。
- 在 drop user 语句的使用中，如果没有明确地给出账户的主机名，则默认为 “%’’。
- 用户的删除不会影响到他们之前所创建的表，索引或其他数据库对象，这是因为 MySQL 并没有记录是谁创建了这些对象。

例如，删除 lisi：

```
drop user lisi@localhost;
```

13.1.3 修改用户账号

```
rename user old_user to new_user [, old_user to new_user ] ...
```

例如，将前面的用户 zhangsan 的名字修改为 jwen：

```
rename user zhangsan@localhost to jwen@localhost;
select user from mysql.user;
```

- rename user 语句用于对原有 MySQL 账户进行重命名。
- 要使用 rename user 语句，必须拥有 MySQL 中 mysql 数据库的 update 权限或 create user 权限。
- 倘若系统中就账户不存在或者新账户已存在，则语句执行会出错。

13.1.4 修改用户口令

```
set password [for user] = ' password'
```

例如：

```
set password for jwen@localhost = '12345';
```

或者使用下面的语句：

```
alter user jwen@localhost identified by '123';
```

- 若是不加上 for 子句，表示修改当前用户的口令。

13.2 账户权限管理

成功创建用户账号后，需要为用户分配适当的访问权限，因为新创建的用户没有访问权限，只能登录 MySQL 服务器，不能执行任何数据库操作。例如，show grants for 语句就能查看某个用户的授权表：

```
show grants for jwen@localhost;
-- 结果:

GRANT USAGE ON *.* TO `jwen`@`localhost`
```

输出结果表明该用户只有一个权限 usage on *.*，表示该用户对任何数据库和任何表都没有权限。

13.2.1 权限的授予

grant 语句的语法格式为：

```
grant
  priv_type [(column_list)] [, priv_type[(column_list)]] ...
  on [object_type] priv_level
  to user_specification [, user_specification] ...
  [require {none | ssl_option [[and] ssl_option]...}]
  [with with_option...]
```

其中 object_type 的格式为：

```
table | function | procedure
```

priv_level 的格式为：

```
* | *.* | db_name.* | db_name.tbl_name | tbl_name | db_name.routine_time
```

user_specification 的格式为：

```
user
[
  identified by 'password'
  | identified with auth_plugin [as 'auth_string']
]
```

with_option 的格式为：

```
grant option
  | max_queries_per_hour count
  | max_updates_per_hour cout
  | max_connections_per_hour count
  | max_user_connections count
```

语法说明如下：

1. priv_type：用于指定授予权限的名称，如 select、update、delete 等操作；
2. 可选项 column_list：用于指定权限要授予给表中那些具体的列；
3. on 子句：用于指定权限授予的对象和级别，如可在 on 关键字后面给出要授予权限的数据库名或表名等；
4. 可选项 object_type：用于指定权限授予的对象类型，包括表、函数和存储过程，分别用关键字 table、function 和 procedure 标识。
5. priv_level：用于指定权限的级别。可以授予的权限有如下几组：
   - 列权限，其和表中的一个具体的列有关。例如，可以使用 update 语句更新表 tb_student 中的 studentName 列的值的权限；

  - 表权限，其和一个具体表中的所有数据相关。例如，可以使用 select 语句查询表 tb_student 的所有数据的权限。
  - 数据库权限，其和一个具体的数据中所有表相关。例如，可以在已有的数据库 db_school 中创建新表的权限。
  - 用户权限，其和 MySQL 中所有的数据库相关。例如，可以删除已有的数据库或者创建一个新的数据库的权限。
- 对应地，在 grant 语句中可以用于指定权限级别的值有如下几类格式：
  - `*`：表示当前数据库中的所有表；
  - `*.*`：表示所有数据库中的所有表；
  - `db_name.*`：表示某个数据库中的所有表；
  - `db_name.tbl_name`：表示某个数据库中的某个表或视图；
  - `tbl_name`：表示某个表或视图；
  - `db_name.routine_time`：表示某个数据库中的某个存储过程或函数。

1. to 子句：用来设定用户的口令，以及指定授予权限的用户 user。若 to 子句中给系统中存在的用户指定口令，则新密码会覆盖原密码；如果权限被授予给一个不存在的用户，MySQL 会自动执行一条 create user 语句创建这个用户，但同时必须为该用户指定口令。由此可见，grant 语句也可以用来创建用户账号。
2. user_specification：to 子句中的具体描述部分，其与 create user 语句汇总的 user_specification 部分一样。
3. with 子句：grant 语句的最后可以使用 with 子句，为可选项，其用于实现权限的转移和限制。

【例如】授予用户 jwen 在数据库 db_school 的表 tb_student 上拥有对 studentNo 和列 studentName 的 select 权限：

```
use db_school;
grant select(studentNo, studentName)
on db_school.tb_student
to jwen@localhost;
```

【例如】当前系统中不存在用户 liming 和 huang，要求创建这两个用户，并为其设置登录口令，同时授予他们在数据库 db_school 的表 tb_student 上拥有 select 和 update 的权限。

```
-- 错误!
grant select, update
on db_school.tb_student
to 'liming'@'localhost' identified by '123',
  'huang'@'localhost' identified by '789';
```

Error!：上面的语句无法正常允许，似乎是最新版本的 MySQL 已经丢弃了这种用法，所以还是先创建用户再授予权限吧！

```
create user if not exists
  liming@localhost identified by '123',
  huang@localhost identified by '123';
```

```
grant select, update
on db_school.tb_student
to liming@localhost, huang@localhost;
```

【例如】授予系统中已存在用户 jwen 可以在数据库中执行所有数据库操作的权限。

```
grant all on db_school.*
to jwen@localhost;
```

【例如】授予用户 jwen 拥有创建用户的权限。

```
grant create user on *.*
to jwen@localhost;
```

grant 语句中 priv_type 的使用说明如下:

1. 授予表时,priv_type 可以指定为以下值:
   - select:授予用户可以使用 select 语句访问特定表的权限;
   - insert:授予用户可以使用 insert 语句向一个特定表中添加数据行的权限;
   - delete:授予用户可以使用 delete 语句从一个特定表中删除数据行的权限;
   - update:授予用户可以使用 update 语句从一个特定表中删除数据行的权限;
   - reference:授予用户可以创建一个外键来参照特定数据表的权限;
   - create:授予用户可以使用特定名字创建一个数据表的权限;
   - alter:授予用户可以使用 alter table 语句修改数据表的权限;
   - index:授予用户可以在表上定义索引的权限;
   - drop:授予用户可以删除数据表的权限;
   - all 或 all privileges:表示所有权限。
2. 授予列权限时,priv_type 可以被指定为以下值:
   - select:授予用户使用 select 语句访问特定数据库中所有表和视图的权限;
   - insert: 授予用户可以使用 insert 语句向特定数据库中所有表添加数据行的权限;
   - delete: 授予用户可以使用 delete 语句删除特定数据库中所有表的数据行的权限;
   - update:授予用户可以使用 update 语句更新特定数据库中的所有数据表的值的权限;
   - references:授予用户可以创建指向特定数据库中的表外键的权限;
   - create:授予用户可以使用 create table 语句在特定数据库中创建新表的权限;
   - alter: 授予用户可以使用 alter table 语句在特定数据库中修改所有数据表的权限;
   - index:授予用户可以在特定数据库中的所有数据表上定义和删除索引的权限;
   - drop:授予用户可以删除特定数据库中所有表和视图的权限;
   - create temporary tables:授予用户可以在特定数据库中创建临时表的权限;

- create view:授予用户可以在特定数据库中创建新的视图的权限;
- show view:授予用户可以查看特定数据库中已有视图的视图定义的权限;
- create routine:授予用户可以为特定数据库创建存储过程和存储函数等的权限;
- alter routine:授予用户可以更新和删除数据库中已有的存储过程和存储函数的权限;
- execute routine:授予用户可以调用特定数据库的存储过程和存储函数的权限;
- lock tables:授予用户可以锁定特定数据库的已有数据表的权限;
- all / all privileges:授予用户所有权限.

3. 最有效率的权限是用户权限。授予用户权限时,priv_type 除了可以指定为授予数据库权限时的所有值之外,还可以是下面的值:
   - create user:授予用户可以创建和删除新用户的权限;
   - show databases:授予用户可以使用 show databases 语句查看所有已有的数据库的定义的权限。

13.2.2 权限的转移和限制

权限的转移和限制可以通过 grant 语句中的 with 子句来实现。1. 转移权限如果 with 子句指定为 with grant option,则表示 to 子句中所有的用户都具有把子句所拥有的权限授予其他用户的权利,而无论那些其他用户是否具有该权限。

【例如】授予当前系统中的 liming 在数据库 db_school 的表 tb_student 上拥有 select 和 update 的权限,并允许其可以将自身的这个权限授予其他用户。

```
grant select, update
on db_school.tb_student
to liming@localhost
with grant option;
```

限制权限如果 with 子句中 with 关键字后面紧跟的是 max_queries_per_hour 等,则该 grant 语句可用于限制权限。**【例如】**授予系统中的用户 huang 在数据库 db_school 的表 tb_student 上每个小时只能处理一条 delete 语句的权限:

书上的代码允许会出错,我用了下面的语句:

```
grant delete
on db_school.tb_student
to huang@localhost;
alter user huang@localhost with max_queries_per_hour 1;
```

13.2.3 权限的撤销

语法格式为:

```
revoke priv_type [(column_list)] [, priv_type[(column_list)]] ...
on [object_type] priv_level
from user [, user] ...
```

```
revoke all privileges, grant option
from user [, user]
```

要使用 revoke 语句,必须具有 mysql 数据库的全局 create user 权限或 update 权限。

【例如】回收 jwen 在数据库 db_school 的表 tb_student 上的 select 权限.

```
revoke select
on db_school.tb_student
    from jwen@localhost;
```

第 14 章 备份和恢复

MySQL 数据库的日常管理中，通常会进行数据库的备份和恢复。

14.1 MySQL 数据库备份和恢复的方法

MySQL 数据库中的备份和恢复组件为存储在 MySQL 中的关键数据提供了重要的保护手段。### 使用 SQL 语句备份和恢复表数据在 MySQL5.5 中，可以使用 select into ... outfile 语句将表数据导出到一个文本文件中进行备份。并可以使用 load data ... infile 语句恢复先去备份的数据。这种方法有一点不足，就是只能导出或导入数据的内容，而不包括表的结构，若表的结构文件损坏，则必须先设法恢复原来表的结构。

14.1.0.1 select into ... outfile 语句

导出备份语句 select into ... outfile 的语法格式为：

```
select * into outfile 'file_name' [character set charset_name] export_options
    | into dumpfile 'file_name'
```

其中，export_options 的格式为：

```
[fields
  [terminated by 'string']
  [[optionally] enclosed by 'char']
  [escaped by 'char']
]
```

语法说明如下：

- 这个语句的作用是把 select 语句选中的所有数据行写入到一个文件中。文件默认在服务器主机上创建，并且文件名不能是已存在的，否则可能将原文件覆盖。如果要将该文件写入到一个特定的位置，则要在文件名前面加上具体的路径。在文件中，导出的数据行会以一定的形式存放，其中空值用\N 表示。
- 导出语句中使用关键字 outfile 时，可以在 export_options 中加入以下两个自选的子句。它们的作用是决定数据行在备份文件中存放的格式：
    1. fields 子句：在 fields 子句中有三个亚子句，分别是 terminated by、[optionally] enclosed by 和 escaped by。如果指定了 fields 子句，则这三个亚子句至少需要指定一个。其中，terminated by 用来指定字段值之间的符号，例如，“terminated by ‘,’ ”，表示指定逗号为两个字段值之间的标志；enclosed by 子句用来指定包裹文件中字符值的符号，例如，“enclosed by ”’“” 表示文件中字符值放在双引号之间。若加上关键字 optionally 则表示所有的值都放在双引号之间；escaped by

子句用来指定转义字符，例如，“ escaped by '*' ”将“ * ”指定为转义字符，取代“\”，如空格将表示为“ *N ”。

2. lines 子句：在 lines 子句中使用 terminated by 指定一个数据行结束的标志，如“lines terminated by ‘?’ ” 表示数据行以问号为结束标志。

如果 fields 和 lines 子句都不指定，则默认声明是下面的子句：

```
fields terminated by '\t' enclosed by '' escaped by '\\'
lines terminated by '\n'
```

- 导出语句中使用的是关键字 dumpfile 而非 outfile 时，导出的备份文件里面所有的数据行都会彼此紧挨着放置，即值和行之间没有任何标记。

14.1.0.2 load data ... infile 语句

导入恢复语句 load data ... infile 语句的语法格式是：

```
load data [low_priority | concurrent] [local] infile 'infile.txt'
  [replace | ignore]
  into table tbl_name
  [fields
    [terminated by 'string']
    [[optionally] enclosed by 'char']
    [escaped by 'char']
  ]
  [lines
    [starting by 'string']
    [terminated by 'string']
  ]
  [ignore number lines]
  [(col_name_or_user_var, ... )]
  [set col_name = expr, ...]
```

语法说明如下：

- low_priority | concurrent：若指定 low_priority，则延迟该语句的执行；若指定 concurrent，则当 load data 正在执行的时候，其他线程可以同时使用该表的数据。
- local：若指定了 local，则文件会在客户主机上的客户端读取，并被发送到服务器。文件会被给予一个完整的路径名称，以指定确切的位置。如果给定的是一个相对的路径名称，则此名称会被理解为相对启动客户端时所在的目录。若没有指定 local，则文件必须位于服务器的主机上，并且被服务器直接读取。与让服务器直接读取相比，使用 local 的速度会略慢些，这是由于文件的内容必须通过客户端发送到服务器上，
- file_name：待导入的数据库备份文件，文件中保存了待载入数据库的所有数据行。输入文件可以手动创建，也可以使用其它的程序创建。导入文件时可以指定文件的绝对路径。若不指定路径，则服务器默认在默认数据库的数据库目录中读取。
- replace | ignore：如果指定 replace，则当导入文件中出现与数据库中原有行相同的唯一关键字时，输入行会替代原有行；如果指定 ignore，则把与原有行有相同的唯一关

键字的输入行跳过。

- tbl_name：指定需要导入数据的表名，该表在数据库中必须存在，表结构必须与导入文件的数据行一致。
- fields 子句：此处的 fields 子句和 select ... into outfile 语句中类似，用于判断字段之间和数据行之间的符号。
- lines 子句：terminated by 亚子句用来指定一行结束的标志；starting by 亚子句则指定一个前缀，导入数据行时，忽略数据行中该前缀和前缀之前的内容。如果某行不包括该前缀，导入数据行时，忽略数据行中的该前缀和前缀之前的内容。如果某行不包括该前缀，则整个数据行被跳过。
- ignore number lines：这个选项可以用于忽略文件的前几行。例如，可以使用 ignore 1 lines 来跳过第一行。
- col_name_or_user_var：如果需要载入一个表的部分列，或者文件中字段值顺序与源表中列的顺序不同时，就必须指定一个列清单，其中可以包含列名或用户变量。例如：

```
load data infile 'backupfile.txt'
  into table backupfile (cust_id, cust_name, cust_address);
```

- set 子句：set 子句可以在导入数据时修改表中列的值。

例如，备份数据库 db_school.tb_student。

```
select * from db_school.tb_student
into outfile '/Users/mr.cheng/sql/backupfile.txt'
fields terminated by ','
optionally enclosed by '"'
lines terminated by '?';
```

这里会遇到一个非常让人头疼的问题，不过幸好最后找到了解决办法。详细解决方法可以参考我的另一篇博客《如何解决 MySQL 导出数据时的权限设置问题?》

然后使用下面的语句将备份数据导入数据库 db_school 中一个和 tb_school 表结构相同的空表 tb_student_copy 中：

```
# 然后使用下面的语句将备份数据导入数据库db_school中一个和tb_school表结构相同
    的空表tb_student_copy中：

load data infile '/Users/mr.cheng/sql/backupfile.txt'
    into table db_school.tb_student_copy
fields terminated by ','
optionally enclosed by '"'
lines terminated by '?';
```

需要注意，上面的语句在我的电脑上直接运行是会出错的，出错的原因是刚刚导出的文件不是 UTF8 编码的，然后我。。。用 Stata 转码的。

在导入数据时需要特别注意，必须根据数据备份文件中数据行的格式来指定判断的符号。例如，在 backupfile.txt 文件中字段值是以逗号隔开的，导入数据时也一定要指定用逗号隔开，即与导出语句相对应。

另外需要注意的是，在多个用户同时使用 MySQL 数据库的情况下，为了得到一个一致的备份，需要在指定的表上使用 lock tables table_name read 语句做一个读锁定，以防止备份的过程中，表被其他用户更新；而当恢复数据时，需要使用 loack tables table_name write 语句做一个写锁定，以防止发生数据冲突。在数据备份或恢复之后需要使用 unlock tables 语句对该表进行解锁。

14.1.1 使用 MySQL 客户端使用程序备份和恢复数据

MySQL 提供了许多免费的客户端实用程序，且存放在 MySQL 安装目录下的 bin 子目录中，这些客户端实用程序可以连接到 MySQL 服务器进行数据库的访问，或者对 MySQL 执行不同的管理任务。其中，mysqldump 和 mysqlimport 就分别是两个常用于实现数据库备份和恢复的实用工具。

14.1.1.1 使用 MySQL 客户端实用程序的方法

首先进入 bin 目录（实际上没有必要的，似乎安装 MySQL 的似乎这个路径就已经被添加到了环境变量中，所以可以全局使用）：

```
cd '/usr/local/mysql-8.0.11-macos10.13-x86_64/bin/'
```

14.1.1.2 使用 mysqldump 备份数据

1. 备份表：

```
mysqldump [options] database [tables] > file_name
```

例如，使用 mysqldump 备份 db_school.tb_student：

```
mysqldump -h localhost -u root -p12345 db_school tb_student > ~/Desktop/file.
    sql
```

然后就会得到一个 sql 文件，运行这个 sql 文件即可恢复数据库。

1. 备份数据库：

   ```
   mysqldump [options] --databases [options] DB1 [DB2 DB3] > file_name
   ```

   例如，备份数据库 ：

   ```
   mysqldump -u root -p12345 --databases 基础数据 > ~/Desktop/基础数据.sql
   ```

2. 备份整个数据库系统：

   ```
   mysqldump [options] --all-databases [options] > file_name
   ```

   例如：

```
mysqldump -u root -p12345 --all-databases > ~/Desktop/alldata.sql
```

需要注意的是，尽管使用 mysqldump 程序可以有效地导出表的结果，但是恢复数据的似乎，倘若所需恢复的数据量很大，备份文件中众多的 SQL 语句会使恢复的效率降低。因此，可以在 mysqldump 命令中使用`--tab=`选项来分开数据和创建表的 SQL 语句。`--tab=`选项会在选项中 “=’’ 后面指定的目录里分别创建存储数据内容的 txt 文件和包含创建表结构的 sql 文件。另外，该选项不能与`--databases`选项或者`--all-databases`选项同时使用，并且 mysqldump 必须运行在服务器主机上。

例如，将数据库 db_school 中所有表的结构和数据分别存放备份：

```
sudo mysqldump -u root -p12345 db_school tb_student  --tab='/Users/mr.cheng/
    sql/'
```

需要注意的是，这个似乎需要导入数据，因此需要导入到 MySQL 具有权限的文件夹 sql 里面（我把这个文件夹的权限分配给了 MySQL）。

14.1.1.3 使用 mysql 命令恢复数据

例如，假设 db_school 数据库遭到破坏，需要使用 db_school.sql 进行恢复：

```
mysqldump -u root -p12345 --databases db_school > ~/Desktop/db_school.sql
mysql -u root -p12345 db_school < ~/Desktop/db_school.sql
```

再例如，只恢复一个表：

```
mysqldump -u root -p12345 db_school tb_student > ~/Desktop/tb_student.sql
mysql -u root -p12345 db_school < ~/Desktop/tb_student.sql
```

14.1.1.4 使用 mysqlimport 程序恢复数据

倘若只是恢复数据表中的数据，可以使用 mysqlimport 程序完成。这个程序提供了 load data . . . infile 语句的一个命令行接口，它发送一个 load data infile 命令到服务器来运作，其大多数选项直接对应 load data . . . file 语句：

```
mysqlimport [options] database textfile ...
```

语法说明如下：

1. 常见的选项有：
    - -d / –d：在导入文本文件之前清空表中的所有数据行；
    - -l / –lock-tables：在处理任何文本之前锁定所有的表，以保证所有的表在服务器上同步，但对于 InnoDB 类型的表这不必进行锁定。
    - –low-priority / –local / –replace / –ignore：分别对应 load data . . . infile 语句中的 low_priority / local / replace / ignore 关键字。

2. textfile: 存储备份数据的文本文件名。使用 mysqlimport 命令恢复数据的时候，mysqlimport 会剥去这个文件的扩展名，并使用它觉得向数据中哪个表导入文件的内容。例如，file.txt / file.sql / file 都会被导入到名为 file 的表中，因此备份文件需要根据需要恢复表命名。另外，在该命令中需要指定备份文件的具体路径，若没有指定，则选取文件的默认位置。即 MySQL 安装目录下面的 data 目录下。

例如：

```
mysqlimport -u root -p12345 --low-priority --replace db_school /Users/mr.
    cheng/sql/tb_student.txt
```

不过由于外键的约束，上面的命令无法执行。

14.2 二进制文件的使用

在 MySQL 的实际操作中，数据库管理员不可能无时无刻地都在备份数据。因此，但数据遭遇丢失或者被破坏的时候，只能恢复已经备份的文件，而在这之后更新的数据就无法恢复了。遭遇到这种情形时，可以考虑使用更新日志，这是因为更新日志可以实时记录下数据库中的修改、插入和删除的 SQL 语句。

在 MySQL5.5 中，更新日志已经被二进制日志取代。二进制日志是一种更加有效的格式，并且是事务安全的方式，包含了更新日志中可用的所有信息，如关于每个更新数据库的语句的执行时间信息，而不包含没有任何修改数据的语句。这些语句以时间的形式保存，记录了数据的更改。

由于二进制日志包含了数据备份后进行的所有更新，因此二进制的主要目的在于数据恢复时最大可能地更新数据库。

14.2.1 开启日志文件

在 `my.cnf` 文件中的 [mysqld] 标签下添加一行：

```
log-bin = /Users/mr.cheng/sql/二进制日志
```

`/Users/mr.cheng/sql/ .txt` 是我设定的日志文件路径和名称。

加入该选项后，重启 MySQL，服务器启动的时候会加载该选项，从而启动二进制日志。如果 filename 中有扩展名，扩展名会被忽略。MySQL 服务器会为每个二进制文件名后面自动添加一个数字编号扩展名，每次启动服务器或刷新日期的时候，都会重新生成一个二进制日志文件，扩展名中的数字编号依次递增。如果未给出文件名，默认为主机名。

14.2.2 使用 mysqlbinlog 工具处理日志

1. 查看二进制日志文件：

例如：

```
sudo mysqlbinlog ~/sql/二进制日志.000001
```

sudo 是因为这个文件夹的权限被分配给了 MySQL。

2. 使用二进制日志恢复数据:

【假如】系统管理员在本周一下午五点下班前,使用 mysqldump 进行了数据库 db_school 的一个完全备份,备份文件为 alldata.sql。接着,从本周一下午 5 点开始启用日志,日志文件中保存了本周一下午 5 点至本周三上午 9 点的所有更改信息,在周三上午九点运行了一条日志刷新语句,即 `flush logs;`,此时数据库自动创建了一个新的日志文件,直至周五上午十点公司数据库服务器系统崩溃。要求恢复到崩溃前的状态。

第一步,使用 sql 文件将数据库恢复到周一下午五点的状态:

```
mysqldump -u root -p12345 db_school < alldata.sql
```

第二步,使用 mysqlbinlog 工具将数据库恢复到周三的状态:

```
mysqlbinlog bin_log.000001 | mysql -u root -p12345
```

最后,使用 mysqlbinlog 将数据库恢复到周五的状态:

```
mysqlbinlog bin_log.000002 | mysql -u root -p12345
```

由于二进制文件会占用很大的硬盘资源,因此需要及时清除没用的二进制日志文件。可以使用下面的这条 sql 语句清除所有的日志文件:

```
reset master;
```

如果需要删除部分日志文件,可以使用 purge master logs 语句来实现:

```
purge {master | binary } logs to 'log_name';
-- 或
purge {master | binary} logs before 'date';
```

第一条语句用于删除指定的日志文件,第二条则用于删除指定日期前的日志文件。

第 15 章 MySQL 数据库的应用编程

MySQL 作为一类中小型关系数据库管理系统，目前已被广泛应用于互联网中各种中小型完整或信息管理系统的开发，其所搭建的应用环境主要有 LAMP 和 WAMP 两者，它们均可使用 PHP 作为于 MySQL 数据库进行交互的服务器端脚本语言。本文主要介绍使用 PHP 进行 MySQL 数据库应用编程的相关知识。

15.1 PHP 概述

PHP 是 Hypertext Preprocessor（超文本预处理器）的递归缩写，目前使用相当广泛，它是一种在服务器端执行的 HTML 文档的脚本语言，语言风格类似于 C 语言，其独特的语法混合了 C、Java、Perl 以及 PHP 自创的新语法。PHP 作为一种服务器端的脚本/编程语言，凭借其简单、面向对象、解释型、高性能、独立于框架、动态、可移植等特点，成为了当前世界上最流行的构建 B/S 模式 Web 应用程序的编程语言之一。PHP 具有强大的功能，其能实现所有的 GGI 的功能，并可提供比一般的 GGI 更快的执行速度，它的多平台特性使其能无缝地运行在 Unix 和 Windows 平台。另外，更为突出的是它对数据库强大的操作能力以及操作的简便性，使其可以方便快捷地操作几乎所有留下的数据库，其中 PHP 搭配 MySQL 成为目前 Web 应用开发的最佳组合。

15.2 PHP 编程的基础

例如，编写一个可以通过浏览器网页显示 “hello world’’ 的 PHP5 程序代码：

```
<html>
<head>
    <title>Hello world!</title>
</head>
<body>
<h1>
    <?php
    $string = "Hello!";
    echo $string;
    ?>
</h1>
</body>
</html>
```

由 PHP 代码生成的的网页代码为：

```
<html>
<head>
    <title>Hello world!</title>
```

```
</head>
<body>
<h1>
    Hello!</h1>
</body>
</html>
```

由此可见，PHP 程序是在 Web 服务器端运行的，且最终会以 HTML 文档的格式输出到客户端/浏览器。从语法上来看，PHP 语言是借鉴 C 语言的语法特征，由 C 语言改进而来的。在 PHP 应用程序编写的过程中，可以混合编写 PHP 和 HTML，即不仅仅可以将 PHP5 代码脚本通过标签 <? php 和?> 嵌入到 Html 中，还可以把 HTML 标签嵌入到 PHP5 的脚本里。

15.3 使用 PHP 进行 MySQL 数据库应用编程

针对不同的应用，PHP 内置了很多函数。为了在 PHP5 程序中实现对 MySQL 数据库的各种操作，可以使用其中的 mysql 函数库。然而，在使用 mysql 函数库访问 MySQL 数据库之前，需要在 PHP 的配置文件 php.ini 中将 “; extension=php_mysql.dll’’ 中的 “;’’ 删除。然后重启 Web 服务器。

通过内置的函数库 mysql，PHP5 程序能够很好地与 MySQL 数据库进行交互。使用这种方式所构建的基于 B/S 模式的 Web 应用程序的工作流程可以描述如下：

1. 在用户计算机的浏览器中通过在地址栏中输入相应的 URI 信息，向网页服务器提出交互请求。
2. 网页服务器收到用户浏览器端的交互请求。
3. 网页服务器根据请求寻找服务器上的网页。
4. Web 应用服务器执行页面内含的 PHP 代码脚本程序。
5. PHP 代码脚本程序通过内置的 MySQL API 函数访问后台 MySQL 数据库服务器。
6. PHP 代码脚本程序取回后台 MySQL 数据库服务器的查询结果。
7. 网页服务器将查询结果以 HTML 文档的格式返回给用户浏览器。

15.3.1 编程步骤

使用 PHP 进行 MySQ 数据库编程的基本步骤如下：

1. 首先建立与 MySQL 数据库编程的基本步骤；
2. 然后选择要对其进行操作的数据库；
3. 再执行相应的数据库操作，包括对数据的添加、删除、修改和查询等。
4. 最后关闭与 MySQL 数据库服务器的连接。

以上各步骤，均是通过 PHP 5 内置函数库 mysql 中相应的函数来实现的。

15.3.2 建立与 MySQL 数据库服务器的连接

在 PHP5 中，可以使用函数 mysql_connect() 和函数 mysql_pconnect() 来建立与 MySQL 数据库服务器的连接。其中函数 mysql_connect() 用于建立非持久连接，而 mysql_pconnect() 用于建立持久连接。

15.3.2.1 使用函数 mysqli_connect() 建立连接

在 PHP5 中，函数 mysql_connect() 函数的语法格式为 (**这个函数在 PHP7 中已被废弃！**)：

```
mysql_connect([servername[, username[, password]]])
```

语法说明如下：

- servername：可选项，为字符串型，用于指定要连接的数据库服务器。默认值为 “localhost:3306”；
- username：可选项，字符串型，用于指定登录服务器所使用的用户名。默认值为拥有服务器进程的用户的名称，如超级用户 root；
- password：可选项，字符串型，用于指定登录数据库服务器所用的密码，默认为空串；
- 函数 mysql_connect() 的返回值为资源句柄型（resource）。若成功执行，则返回一个连接标识号；否则返回 false。

在 PHP 程序中，通常将该函数返回的连接标识号存储在某个变量中，以备 PHP 程序使用。实际上，在后续其他有关操作 MySQL 数据库的函数中，一般都需要指定相应的连接标识号作为该函数的实参。

例如，编写一个数据库服务器的连接示例程序 connect.php，连接 MySQL 数据库并使用变量 $con 保存连接的结果。

```
<?php
$con = mysqli_connect("localhost", "root", "12345");
if(!$con)  {
    echo "连接失败! <br>";
    echo "错误编号:".mysqli_errno()."<br>";
    echo "错误信息:".mysqli_error()."<br>";
}
else{
    echo "连接成功";
}
```

注意：在 PHP7 中，mysql_connect 已被废弃，所以需要使用 mysqli_connect。上面的代码运行环境为 PHP7.2+MySQL8.0。直接运行会出现错误，解决办法参考php7 连接 mysql8

在 PHP 中，一切非 0 值会被认为是 true，而数值 0 则被当作逻辑值 false。`mysqli_connect()` 函数执行成功后，实际上返回的标识号是一个非 0 值，即被当作逻辑值 true 来处理。因而判断是否成功连接，只需要判断返回值即可。

如果连接失败,则可进一步调用 PHP 中的函数 `mysqli_errno()` 和 `mysqli_error()`,以获取相应的错误编号和错误提示信息。当一个 MySQL 函数成功执行后,函数 `mysqli_errno()` 和 `mysqli_error()` 分别会返回数值 0 和空字符串,因此,这两个函数也可以用于判断函数 `mysqli_connect()` 或者其他 MySQL 函数的执行情况。

15.4 补充部分:

PHP 连接 MySQL PHP5 及以上版本建议使用以下方式连接 MySQL:

- MySQLi extension ("i'' 意为 improved)
- PDO(PHP Data Objects)

在 PHP 早期版本中使用 MySQL 扩展,但是该扩展在 2012 年开始不建议使用。

该用 MySQLi 还是 PDO? + ''习惯哪个就用哪个!" + MySQLi 和 PDO 各有优势:

- PDO 应用在 12 种不同数据库中,而 MySQLi 只针对 MySQL 数据库;
- 所以如果你的项目需要在多种数据库中切换,建议使用 PDO,这样你只需要修改连接字符串和部分查询语句即可。使用 MySQLi,如果不同数据库,你需要重新编写所有的代码,包括查询。
- 两者都是面向对象,但是 MySQL 还提供了 API 接口。
- 两者都支持预处理语句,预处理语句可以防止 SQL 注入,对于 web 项目的安全性非常重要的。

15.4.1 面向对象的 MySQL 数据库连接

```
<?php
$servername = "localhost";
$username = "root";
$password = "12345";
$conn = new mysqli($servername, $username, $password)
if ($conn -> connect_error){
  die("连接失败:
".$conn -> connect_error);
}
echo "连接成功! ";
?>
```

15.4.2 面向过程的 MySQL 数据库连接

```
<?php
$servername = "localhost";
$username = "root";
$password = "12345";
$conn = mysqli_connect($servername, $username, $password);
if (!$conn) {
```

```
  die("连接失败:
".mysqli_connect_error());
}
echo "连接成功! ";
?>
```

15.4.3 PDO 连接 MySQL

```
<?
$servername = "localhost";
$username = "root";
$password = "12345";
try{
  $conn = new PDO("mysql:host = $servername;", $username, $password);
  echo "连接成功! ";
}
catch(PDOException $e)
{
  echo $e -> getMessage();
}
?>
```

15.4.4 关闭连接

15.4.4.1 面向对象的 MySQLi

```
$conn -> close();
```

15.4.4.2 面向过程的 MySQLi

```
mysqli_close($conn);
```

15.4.4.3 PDO

```
$conn = null;
```

15.5 选择数据库

```
<?php
/**
 * 选择数据库
 * Created by PhpStorm.
```

```
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 21:41
 */
$con = mysqli_connect("localhost:3306", "root", "12345");
if(mysqli_errno()){
    echo "<font color='red'>数据库服务器连接失败! </font><br>";
    die();
}
mysqli_select_db($con, "db_school");
if(mysqli_errno()){
    echo "<font color='red'>数据库选择失败</font><br>";
    die();
}
else{
    echo "<font color='green'>数据库选择成功!</font><br>";
}
?>
```

15.6 执行数据库操作

15.6.1 数据的添加

```
<?php
/**
 * 执行数据库操作
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 20:40
 */
$con = mysqli_connect("localhost:3306", "root", "12345")
    or die("数据库连接失败! <br>");
mysqli_select_db($con,"db_school") or die("<font color='#ff070f'>数据库选择失
    败! </font><br>");
mysqli_query($con, "set names 'utf8'");
$sql = "insert into db_school.tb_student(studentNo, studentName, sex,
    birthday, native, nation, classNo)
    values
        ('2013122102', '张五勇', '男', '1997-12-11','山西', '汉', 'AC1301')";
if (mysqli_query($con, $sql)){
    echo "<br><font color='#04ff06'>学生插入成功!</font>";
}
else{
    echo "<br><font color='#ff070f'>学生添加失败!</font>";
}
```

```
?>
```

15.6.2 数据的修改

为了方便我把连接数据库部分的代码单独拿出来存为 con.php 文件：

```
<?php
/**
 * 连接mysql数据库
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 22:22
 */
$host = "localhost:3306";
$user = "root";
$password = "12345";
$con = mysqli_connect($host, $user, $password);
?>
```

然后就可以在其他的程序中调用它了！

```
<?php
/**
 * 数据的修改
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 22:21
 */
require "con.php";
mysqli_select_db($con, "db_school") or die("数据库选择失败！<br>");
mysqli_query($con, "set names 'utf8'");
$sql = "update db_school.tb_student set native = '广州'";
$sql = $sql."where studentName = '张晓勇'";
if(mysqli_query($con, $sql)){
    echo "修改成功！";
}
else{
    echo "修改失败！";
}
?>
```

15.6.3 数据的删除

```
<?php
/**
```

```
 * 删除
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 23:08
 */
require "con.php";
mysqli_select_db($con, "db_school");
mysqli_query($con, "set names utf8");
$sql = "delete from tb_student";
$sql = $sql." where studentName = '张五勇'";
if (mysqli_query($con, $sql))
    echo "学生删除成功！";
else
    echo "学生删除失败！";
?>
```

15.6.4 数据的查询

在 PHP 程序中，可以将 MySQL 中用于数据检索的 select 语句置于函数 mysqli_query() 中，实现在选定的数据库中查询所要的数据。此时 mysqli_query() 被成功执行时返回的值不再是一个逻辑值 true，而是一个资源句柄型的结果标识符。结果标识符也被称为结果集，代表了相应查询语句的查询结果。每个结果集中都有一个记录指针，所指向的记录即为当前记录。在初始状态下，结果集的当前记录就是第一条记录。为了灵活地处理结果集中的相关记录，PHP 提供了一系列的处理函数，包括结果集中记录的读取、指针的定位以及记录集的释放等。

15.6.4.1 读取结果集中的记录

在 PHP5 中，可以使用下列函数来读取结果集中的函数：

```
mysql_fetch_array(data[, array_type])
mysql_fetch_row(data)
mysql_fetch_assoc(data)
```

语法说明如下：

- data：为资源句柄型，用于指定要使用的数据指针。该数据指针可以指向函数 mysql_query() 产生的结果集，即结果标志符。
- array_type：可选项，为整型，用于指定函数返回值的形式，其有效取值为 PHP 常量 MYSQL_NUM(表示数字数组)、MYSQL_ASSOC(表示关联数组) 或 MYSQL_BOTH(表示同时产生关联数组和数字数组)。其默认值为 MYSQL_BOTH。
- 三个函数被成功执行后，其返回值均为数组类型。若成功，即读取到当前记录，则返回一个由结果集当前记录所生成的数据，其中每个字段的值会保存在相应的索引之中，并自动将记录指针指向下一个记录。若失败，即没有读取到记录，则返回

FALSE。在使用函数 mysql_fetch_array() 时，若以常量 MYSQL_NUM 作为第二个参数，其功能与函数 mysql_fetch_row() 的功能是一样的，所返回的数据均为数字索引方式的数组，只能以相应的序号 (从 0 开始) 作为元素的下标进行访问；若以常量 MYSQL_ASSOC 作为第二个参数，则其功能与 mysql_fetch_assoc 的功能一样，所返回的数组作为关联索引方式的数组，只能以相应的字段名（如果指定了别名，则为相应的别名）作为元素的下标进行访问；若未指定第二个参数，或以 MYSQL_BOTH 作为第二个参数，则返回的数组以数字索引方式与关联索引方式的数组，既能以序号为元素的下标进行访问，又能以字段名作为元素的下标进行访问。由此可见，mysql_fetch_array() 完全包含了 mysql_fetch_row() 和 mysql_fetch_assco() 的功能。因此在实际使用中更加常用。

例如，编写一个检索数据的 PHP 示例程序 select.php，要求在数据库中查询学号为“2013110101’’ 的学生的姓名。

```
<?php
/**
 * 在数据库中查询学号为"2013110101"的学生的姓名
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 11:43
 */
require "con.php";
mysqli_select_db($con, "db_school") or die("数据库选择失败! <br>");
mysqli_query($con, "set names utf8");
$sql = "select studentName from tb_student";
$sdl = $sql." where studentNo = '2013110101'";
$result = mysqli_query($con, $sql);
if ($result){
    echo "<font color='green'>学生查询成功! </font><br>";
    $array = mysqli_fetch_array($result, MYSQLI_NUM);
    if($array){
        echo "读到学生信息! <br>";
        echo "所要查询学生的姓名是:
".$array[0];
    }
    else echo "没有找到该学生! <br>";
}
else echo "<font color='red'>学生查询失败! </font><br>";
?>
```

15.6.4.2 读取结果集中的记录数

在 PHP5 中，可以使用 mysql_num_rows() 来读取结果集中的记录数，即数据集的行数。其语法格式为：

```
mysql_num_rows(data)
```

例如,查询女学生的人数:

```
<?php
/**
 * 查询女学生的人数:

 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 12:43
 */
require "con.php";
mysqli_select_db($con, "db_school");
mysqli_query($con, "set names utf8");
$sql = "select * from tb_student";
$sql = $sql." where sex = '女'";
$result = mysqli_query($con, $sql);
if($result){
    echo "<font color='green'>查询成功! </font><br>";
    $num = mysqli_num_rows($result);
    echo "数据库db_school中的女学生数为:
".$num."位";
}
else echo "<font color='red'>查询失败! </font>";
```

15.6.4.3 读取指定记录号的记录

在 PHP5 中,可以使用 mysql_data_seek() 在结果集中随意移动记录的指针,也就是将记录指针直接指向某个记录。其语法格式为:

```
mysql_data_seek(data, row)
```

该函数的返回值为布尔型。例如,编写一个读取指定结果集中记录号的 PHP 程序,要求查询第三位女生的姓名:

```
<?php
/**
 * 查询第三位女生的姓名
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 12:52
 */
require "con.php";
mysqli_select_db($con, "db_school");
```

```
mysqli_query($con, 'set names utf8');
$sql = "select * from tb_student";
$sql = $sql." where sex = '女'";
$result = mysqli_query($con, $sql);
if($result){
    echo "<font color='green'>查询成功! </font><br>";
    if(mysqli_data_seek($result, 2)){
        $array = mysqli_fetch_array($result, MYSQLI_NUM);
        echo "数据库db_school中第3位女生是:
".$array[1];
    }
    else echo "<font color='red'>记录定位失败! </font>";
}
else echo "<font color='red'>查询失败! </font>";
?>
```

15.7 关闭数据库服务器的连接

对 MySQL 数据库的操作执行完毕后，应当及时关闭与 MySQL 数据库服务器的连接，以释放其所占用的系统资源。在 PHP5 中，可以使用函数 mysql_close() 来关闭由函数 mysql_connect() 所建立的与 MySQL 数据库服务器的非持久连接。语法格式为:

```
mysql_close([connection])
```

该函数的返回值为布尔型。例如，编写一个关闭数据库连接的程序:

首先我把 con.php 稍稍修改来一下:

```
<?php
/**
 * 连接mysql数据库
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/6
 * Time: 22:22
 */
$host = "localhost:3306";
$user = "root";
$password = "12345";
$con = mysqli_connect($host, $user, $password);
if($con){
    echo "<font color='green'>连接成功! </font><br>";
}
else echo "<font color='red'>连接失败! </font>";
?>
```

然后在编写关闭程序 close.php:

```php
<?php
/**
 * 关闭连接
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 13:02
 */
require "con.php";
$close = mysqli_close($con);
if($close) echo "关闭完成! ";
else "关闭失败! ";
```

第 16 章 开发实例

本文介绍如何使用 PHP 语言开发一个基于 B/S 结构的简单实例系统——--学生成绩管理系统。

16.1 数据库的设计与实现

这部分需要创建四个表：

班级表、学生表、课程表和成绩表。

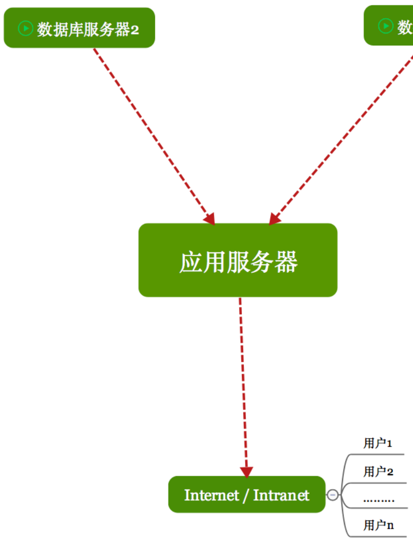
```
drop database if exists db_school;
create database db_school;

# 班级表
use db_school;
create table tb_class
(
  classNo char(6) not null primary key,
  className varchar(20) not null,
  department varchar(30) not null,
  grade smallint,
  classNum tinyint,
  constraint UQ_class unique (className)
)engine = InnoDB;

# 学生表
use db_school;
create table tb_student
(
  studentNo char(10) not null,
  studentName varchar(20) not null,
  sex char(2) not null,
  birthday date,
  native varchar(20),
  nation varchar(10) default '汉',
  classNo char(6),
  constraint PK_student primary key (studentNo),
  constraint FK_student foreign key (classNo)
    references tb_class(classNo)
)engine = InnoDB;

# 课程表tb_course的实现
use db_school;
create table tb_course
```

```
(
  courseNo char(6) not null,
  courseName varchar(20) not null,
  credit int not null,
  courseHour int not null,
  Iterm char(2),
  priorCourse char(6),
  constraint PK_course primary key (courseNo),
  constraint FK_course foreign key (priorCourse)
    references tb_course(courseNo),
  constraint CK_course check (credit = courseHour / 16)
)engine = InnoDB;

# 成绩表
use db_school;
create table tb_score
(
  studentNo char(10) not null,
  courseNo char(6) not null,
  score float check (score >= 0 and score <= 100),
  constraint PK_score primary key (studentNo, courseNo),
  constraint FK_score1 foreign key (studentNo)
    references tb_student(studentNo),
  constraint FK_score2 foreign key (courseNo)
    references tb_course(courseNo)
)engine = InnoDB;

# 插入数据
insert into db_school.tb_class
    values
      ('AC1301', '会计13-1班', '会计学院', 2013, 35),
      ('AC1302', '会计13-2班', '会计学院', 2013, 35),
      ('CS1401', '计算机14-1班', '计算机学院', 2014, 35),
      ('IS1301', '信息系统13-1班', '信息学院', 2013, null),
      ('IS1401', '信息系统14-1班', '信息学院', null, 30);

insert into db_school.tb_student
    values
      ('2013110101', '张晓勇', '男', '1997-12-11', '山西', default, 'AC1301')
        ,
      ('2013110103', '王一敏', '女', '1996-03-25', '河北', default, 'AC1301')
        ,
      ('2013110201', '江山', '女', '1996-03-25', '内蒙古', '锡伯', 'AC1302'),
      ('2013110202', '李明', '男', '1996-01-14', '广西', '壮', 'AC1302'),
      ('2013310101', '黄菊', '女', '1995-09-30', '北京', default, 'IS1301'),
      ('2013310103', '吴昊', '男', '1995-11-08', '河北', default, 'IS1301'),
      ('2014210101', '刘涛', '男', '1997-04-03', '湖南', '侗', 'CS1401'),
```

```
        ('2014210102', '郭志坚', '男', '1997-02-21', '上海', default, 'CS1401')
            ,
        ('2014310101', '王林', '男', '1996-10-09', '河南', default, 'IS1401'),
        ('2014310102', '李怡然', '女', '1996-12-31', '辽宁', default, 'IS1401')
            ;

insert into db_school.tb_course
    values
        ('11003', '管理学', 2, 32, 2, null),
        ('11005', '会计学', 3, 48, 3, null),
        ('21001', '计算机基础', 3, 48, 1, null),
        ('21002', 'OFFICE高级应用', 3, 48, 2, '21001'),
        ('21004', '程序设计', 4, 64, 2, '21001'),
        ('21005', '数据库', 4, 64, 4, '21004'),
        ('21006', '操作系统', 4, 64, 5, '21001'),
        ('31001', '管理信息系统', 3, 48, 3, 21004),
        ('31002', '信息系统_分析与设计', 2, 32, 4, '31001'),
        ('31005', '项目管理', 3, 48, 5, '31001');

insert into db_school.tb_score
    values
        ('2013110101', '11003', 90),
        ('2013110101', '21001', 86),
        ('2013110103', '11003', 89),
        ('2013110103', '21001', 88),
        ('2013110201', '11003', 78),
        ('2013110201', '21001', 92),
        ('2013110202', '11003', 82),
        ('2013110202', '21001', 85),
        ('2013310101', '21004', 83),
        ('2013310101', '31002', 68),
        ('2013310103', '21004', 80),
        ('2013310103', '31002', 76),
        ('2014210101', '21002', 93),
        ('2014210101', '21004', 89),
        ('2014210102', '21002', 95),
        ('2014210102', '21004', 88),
        ('2014310101', '21001', 79),
        ('2014310101', '21004', 80),
        ('2014310102', '21001', 91),
        ('2014310102', '21004', 87);
```

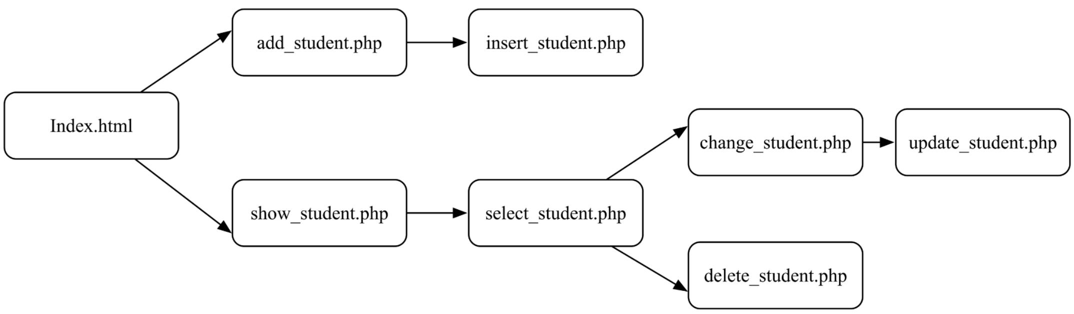
16.2 应用系统的编程

由于本实例系统的最终运行和应用都是基于 B/S 结构，因此本实例系统的开发与实现将采用下图所示的三层软件架构：



上图所示的三层软件体系架构由表示成、应用层和数据层煮成。其中，表示层是本实例系统的用户接口，具体表现为 Web 页面，其主要使用 HTML 标签语言实现；应用层是本实例系统的功能层，表现为应用服务器，位于表示层和数据层之间，主要负责具体的业务逻辑处理，以及表示层、数据层的信息交互，其所处理的各种业务逻辑主要由 PHP 语言编写的动态脚本来实现；数据层位于本案例系统的最底层，具体表现为 MySQL 数据库服务器，其主要通过 SQL 数据库操作语言，负责对 MySQL 数据库中的数据进行读写管理，以及更新的检索，并与应用层实现数据交互。

本案例所需的代码文件之间的链接关系如下图：



16.2.1 实例系统的主页面设计与实现

```
<html>
```

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>学生成绩管理系统实例</title>
</head>

<body>
<h2>学生管理系统</h2>
<h3>学生管理</h3>
<a href="add_user.php">添加学生</a><br />
<a href="show_user.pnh">查看学生</a>
<h3>班级管理</h3>
<a href="add_dept.php">添加班级</a><br />
<a href="show_dept.php">查看班级</a>
<h3>课程管理</h3>
<a href="add_usergroup.php">添加课程</a><br />
<a href="show_usergroup.php">查看课程</a>
<h3>成绩管理</h3>
<a href="add_fun.php">添加成绩</a><br />
<a href="show_fun.php">查看成绩</a>
</body>
</html>
```

16.2.2 公共代码模块的设计与实现

连接数据库、选择数据库和设置字符集三个部分为公共部分，可以单独写给 common.php:

```
<?php
/**
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 15:11
 */
$con = mysqli_connect("localhost:3306", "root", "12345")
    or die("<font color='red'>数据库服务器连接失败! </font><br>");
mysqli_select_db($con, "db_school")
    or die("<font color='red'>数据库选择失败! </font><br>");
mysqli_query($con, "set names utf8");
?>
```

效果:

学生管理系统

学生管理

添加学生

查看学生

班级管理

添加班级

查看班级

课程管理

添加课程

查看课程

成绩管理

添加成绩

查看成绩

16.2.3 添加学生页面设计与实现

通过该页面，可以向学生管理系统添加一位新的学生个人信息。添加学生的 Web 页面文件 add_student.php 的实现代码如下：

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset='UTF-8'">
    <title>添加学生</title>
</head>
<?php require "common.php"; ?>
```

```
<body>
<h3>添加学生</h3>
<form id = "add_student" name = "add_student" method = "post" action="
    insert_student.php">
    学生学号:

<input type="text" name="studentNo" /><br />
    学生姓名:

<input type="text" name="studentName" /><br />
    学生性别:

<input type="text" name="sex" /><br />
    学生年龄:

<input type="text" name="birthday" /><br />
    学生籍贯:

<input type="text" name="native" /><br />
    学生民族:

<input type="text" name="nation" /><br />
    所属班级:

<select name="classNo">
        <?php
        $sql = "select * from tb_class";
        $result = mysqli_query($con, $sql);
        while($rows = mysqli_fetch_row($result)){
            echo "<option value=".$rows[0].">".$rows[1]."</option>";
        }
        ?>
    </select><br/>
    <input type="submit" value="添加" />
</form>
</body>
</html>
```

效果:

16.2.4 插入学生部分代码的实现

insert_student.php 部分的代码:

```
<?php
/**
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 15:42
 */
require "common.php";
$studentNo = $_POST['studentNo'];
$studentName = $_POST['studentName'];
$sex = $_POST['sex'];
$birthday = $_POST['birthday'];
$native = $_POST['native'];
$nation = $_POST['nation'];
$classNo = $_POST['classNo'];
$sql = "insert into tb_student(studentNo, studentName, sex, birthday, native,
     nation, classNo)";
$sql = $sql." values('".$studentNo."','".$studentName."','".$sex.
    "','".$birthday."','".$native."','".$nation."','".$classNo."')";
if(mysqli_query($con, $sql)){
    echo "学生添加成功！<br>";
}
else echo "学生添加失败！<br>";
?>
```

把我添加进去试试：

提交之后即可看到 。

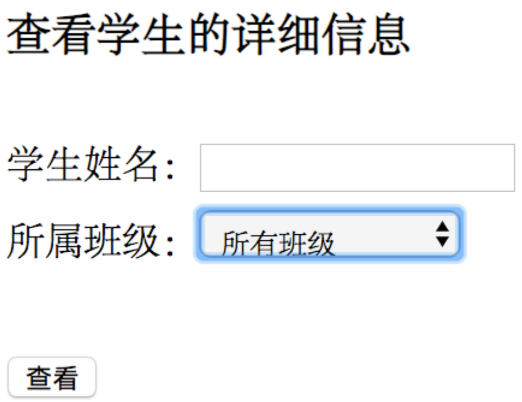
16.2.5 查看学生的页面设计与实现

通过该页面可以通过指定学生姓名或学生所属班级来查看该学生的全部个人信息。查看学生的 Web 页面文件 show_student.php 的实现代码描述如下：

```
<html>
<head>
    <title>查看学生的详细信息</title>
    <meta http-equiv="content-type" content="text/html; charset = 'UTF-8'">
</head>

<body>
<h3>查看学生的详细信息</h3>
<form id="show_student" name="show_student" method="post" action="
    select_student.php">
    学生姓名:

<input type="text" name="show_student_name" /><br />
    所属班级:

<select name="show_student_class">
        <option value = 0>所有班级</option>
        <?php
        require "common.php";
```

```
        $sql = "Select * from tb_class";
        $result = mysqli_query($con, $sql);
        while($rows = mysqli_fetch_row($result)){
            echo "<option values = ".$rows[0].">".$rows[1]."</option>";
        }
        ?>
    </select><br>
    <br>
    <input type="submit" value="查看"/>
</form>
</body>
</html>
```

效果:



select_student.php 部分的代码为:

```
<?php
/**
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 16:20
 */
require "common.php";
$studentName = trim($_POST['show_student_name']);
$classNo = trim($_POST['show_student_class']);
$sql = "select * from tb_student where studentName='";
if($classNo==0)
    $sql = $sql.$studentName."'";
else
    $sql = $sql.$studentName."' and classNo=".$classNo;
$result = mysqli_query($con, $sql);
$array = mysqli_fetch_array($result, MYSQLI_NUM);
$studentNo = $array[0];
$class_sql = "select * from tb_class where classNo=";
$class_sql = $class_sql.$classNo;
```

```
$class_result = mysqli_query($con, $class_sql);
$class_array = mysqli_fetch_array($class_result, MYSQLI_NUM);
$class_name = $class_array[1];
echo "学生学号:

";
echo $array[0];
echo "<br>";
echo "学生姓名:

";
echo $array[1];
echo "<br>";
echo "学生性别:

";
echo $array[2];
echo "<br>";
echo "出生日期:

";
echo $array[3];
echo "<br>";
echo "籍贯:

";
echo $array[4];
echo "<br>";
echo "民族:

";
echo $array[5];
echo "<br>";
echo "所属班级:

";
echo $class_name;
echo "<br>";
?>
<a href="<?php echo 'change_student.php?num='.$studentNo ?>">修改学生信息</a>
<br/>
<a href="<?php echo 'delete_student.php?num='.$studentNo ?>">删除这个调皮的学
    生! </a>
```

学生学号：2012110101

学生姓名：程振兴

学生性别：男

出生日期：1996-10-01

籍贯：安徽

民族：汉

所属班级：会计13-1班

修改学生信息

删除这个调皮的学生！

16.2.6 修改学生的页面设计与实现

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset='UTF-8'">
    <title>修改学生</title>
</head>
<body>
<h3>修改学生信息</h3>
<?php
require_once "common.php";
$studentNo = trim($_GET['num']);
echo "<form id='add_student' name='add_student' method='post' action='
    update_student.php?studentNo=$studentNo'>";
echo "<input type='text' name='studentNo' value='$studentNo' /><br>";
echo "学生姓名:

<input type='text' name='studentName'><br>";
echo "学生性别:

<input type='text' name='sex'><br>";
echo "出生日期:

<input type='text' name='birthday'><br>";
echo "学生籍贯:

<input type='text' name='native'><br>";
```

```
echo "学生民族:

<input type='text' name='nation'><br>";
echo "所属班级:

<select name='classNo'>";
$sql = "select * from tb_class";
$result = mysqli_query($con, $sql);
while($rows = mysqli_fetch_row($result)){
    echo "<option value=".$rows[0].">".$rows[1]."</option>";
}
echo "</select><br>";
echo "<input type='submit' value='确认修改该学生的信息'>";
echo "</form>";
?>
</body>
</html>
```

update_student.php 部分的代码:

```
<?php
/**
 * Created by PhpStorm.
 * User: mr.cheng
 * Date: 2018/9/7
 * Time: 16:57
 */
require_once "common.php";
$studentNo = trim($_GET['studentNo']);
$studentName = trim($_POST['studentName']);
$classNo = trim($_POST['classNo']);
$sex = trim($_POST['sex']);
$birthday = trim($_POST['birthday']);
$native = trim($_POST['native']);
$nation = trim($_POST['nation']);
$sql = "update tb_student set studentName ='".$studentName."', sex = '".$sex.
    "', birthday ='".$birthday."', native ='".$native."', nation='".$nation.
    "', classNo='".$classNo."' where studentNo='";
$sql = $sql.$studentNo."'";
if(mysqli_query($con, $sql))
    echo "学生信息修改成功! ";
else
    echo "学生信息修改失败! ";

?>
```

16.2.7 删除学生的底层代码实现

```
<?php require_once "common.php"; ?>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset='UTF-8'">
    <title>删除学生</title>
</head>
<body>
<?php
$studentNo = trim($_GET['num']);
$sql = "delete from tb_student where studentNo='";
$sql = $sql.$studentNo."'";
if(mysqli_query($con, $sql))
    echo "学生删除成功! <br>";
else
    echo "学生删除失败! <br>";
?>
</body>
</html>
```

终于基本抄完了这本书!

第 17 章 终于解决了 R 连接 MySQL 数据库的问题!

本文 R 脚本:R 数据库导入 MySQL.R 最开始接触 MySQL 的时候就尝试过使用 R 连接 MySQL,但是一直没有成功,主要是没能实现直接将数据框写入 MySQL,`dbWriteTable()` 函数在我的电脑上不好使,所以我不得不直接使用 MySQL 语句写入数据。

17.1 先修改一下我的 MySQL 的登陆密码

之前的那个密码比较敏感,所以不能随便公开,所以得改一个随意的。打开终端输入:

```
mysql -u root -p
-- 会提示输入密码, 输入之前的老密码即可
-- 修改密码为 ' 12345 '
ALTER USER 'root'@'localhost' IDENTIFIED BY '12345';
-- Ctrl+D退出MySQL再重新登陆, 密码设置成功
```

17.2 创建连接

```
library(RMySQL)
# messyr 包已经弃用
library(messyr)
engine = dbConnect(RMySQL::MySQL(), dbname = '交易数据', username = 'root',
    password = '12345', host = 'localhost', port = 3306)
```

上面的交易数据是我的一个数据库。里面存放在一些之前扔进去的数据集。

17.3 查看所选数据库里面的数据

```
dbListTables(engine)
 [1] "000001大单交易数据"          "002337前复权数据"
 [3] "002337后复权数据"            "600848-15K"
 [5] "600848-30K"                 "600848-5K"
 [7] "600848-60K"                 "600848分笔数据"
 [9] "600848周K"                  "600848日K"
[11] "600848月K"                  "601333当日分笔数据"
[13] "个股信息"                   "全部个股当日数据"
[15] "沪深300指数10月份日线行情"    "浦发银行5分钟行情"
[17] "浦发银行前复权日线行情"        "浦发银行后复权周线行情"
[19] "隔夜SHIBOR"                 "鹏华银行分级B的60分钟行情"
```

17.4 从数据库中读取数据存储为数据框

```
ir = dbReadTable(engine, "002337前复权数据")
```

17.5 创建一个数据框

ibor() 函数默认下载的是隔夜 SHIBOR 的数据。

```
# ibor() 函数已经弃用，可以尝试使用其它的数据框
df = ibor()
```

17.6 将数据框插入 MySQL 数据库

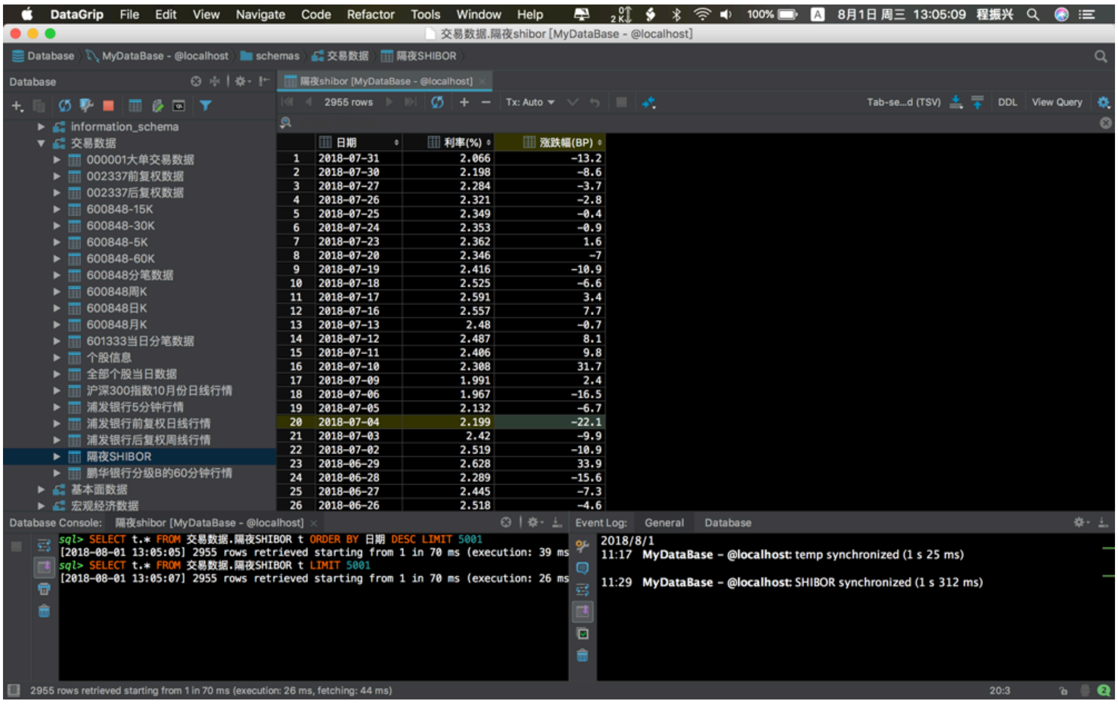
因为 dbWriteTable() 函数在我的电脑上总是出错，所以我不得不使用 MySQL 语句插入表格：

```
# 需要把日期加上单引号才能写入为日期格式
df$日期 = paste0('\'', df$日期,'\'')
# 设置变量名称的编码格式
dbSendQuery(engine,'SET NAMES utf8')
# 删除已有表格
dbSendQuery(engine, "drop table 隔夜SHIBOR")
# 创建表格
dbSendQuery(engine, "CREATE TABLE 隔夜SHIBOR(
             日期 DATE,
             `利率(%)` DOUBLE,
             `涨跌幅(BP)` DOUBLE
             )")
# 插入数据
strSQL <- paste(
  'insert into 隔夜SHIBOR(`日期`, `利率(%)`, `涨跌幅(BP)`) values',
  paste(sprintf("(%s,'%f', %f )", df$日期, df$`利率(%)`, df$`涨跌幅(BP)`),
    collapse=', '),
  sep = ' '
)
dbSendQuery(engine, strSQL)
```

17.7 关闭连接

```
dbDisconnect(engine)
```

17.8 插入效果



17.9 创建拆借利率数据库

既然已经解决了这个问题，那么下一步就是赶紧把拆借利率数据存到数据库里面方便调用啦！首先我写了个函数方便重复执行存入数据库的操作：

```
rm(list = ls())
library(RMySQL)
library(messyr)
engine = dbConnect(RMySQL::MySQL(), dbname = '拆借利率', username = 'root',
    password = '12345', host = 'localhost', port = 3306)
iborpro_mysql <- function(iterm = "SHIBOR-CNY-ON", name = "隔夜SHIBOR"){
  df = iborpro(iterm = iterm)
  df$日期 = paste0('\'', df$日期,'\'')
  sql2 <- paste0(
    "CREATE TABLE ",
    name,
    "(日期 DATE,
    `利率(%)` DOUBLE,
    `涨跌幅(BP)` DOUBLE
  )", sep = ' '
  )
  dbSendQuery(engine, sql2)
  sql3 <- paste0(
  "insert into ", name, "(`日期`, `利率(%)`, `涨跌幅(BP)`) values",
  paste(sprintf("(%s,'%f', %f )", df$日期, df$`利率(%)`, df$`涨跌幅(BP)`),
```

```
    collapse=', ')
  )
  dbSendQuery(engine, sql3)
}
```

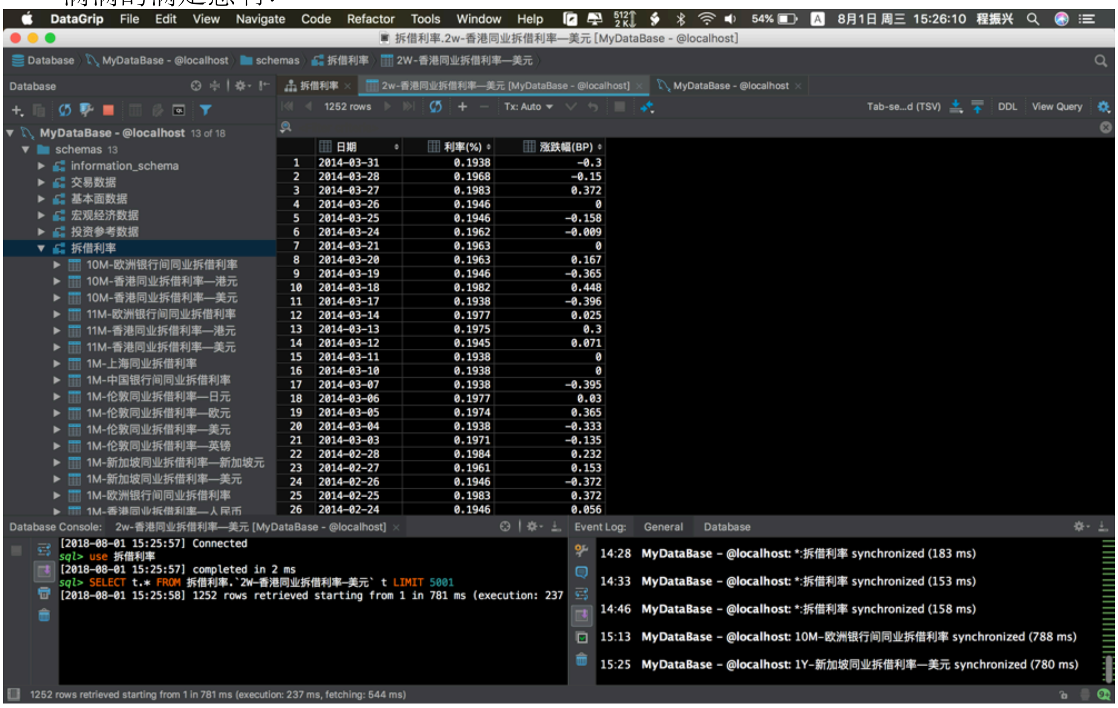
然后把下面的这些繁复的代码运行一遍,数据库就装满啦!实际上我现在一直在担心东方财富网把我的 IP 封了。

```
iborpro_mysql('SHIBOR-CNY-ON', '`ON上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-1W', '`1W上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-2W', '`2W上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-1M', '`1M上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-3M', '`3M上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-6M', '`6M上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-9M', '`9M上海同业拆借利率`')
iborpro_mysql('SHIBOR-CNY-1Y', '`1Y上海同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-ON', '`ON中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-1W', '`1W中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-2W', '`2W中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-3W', '`3W中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-1M', '`1M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-2M', '`2M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-3M', '`3M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-4M', '`4M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-6M', '`6M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-9M', '`9M中国银行间同业拆借利率`')
iborpro_mysql('CHIBOR-CNY-1Y', '`1Y中国银行间同业拆借利率`')
iborpro_mysql('LIBOR-GBP-ON', '`ON伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-GBP-1W', '`1W伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-GBP-1M', '`1M伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-GBP-2M', '`2M伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-GBP-3M', '`3M伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-GBP-8M', '`8M伦敦同业拆借利率英镑`')
iborpro_mysql('LIBOR-USD-ON', '`ON伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-USD-1W', '`1W伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-USD-1M', '`1M伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-USD-2M', '`2M伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-USD-3M', '`3M伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-USD-8M', '`8M伦敦同业拆借利率美元`')
iborpro_mysql('LIBOR-EUR-ON', '`ON伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-EUR-1W', '`1W伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-EUR-1M', '`1M伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-EUR-2M', '`2M伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-EUR-3M', '`3M伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-EUR-8M', '`8M伦敦同业拆借利率欧元`')
iborpro_mysql('LIBOR-JPY-ON', '`ON伦敦同业拆借利率日元`')
iborpro_mysql('LIBOR-JPY-1W', '`1W伦敦同业拆借利率日元`')
iborpro_mysql('LIBOR-JPY-1M', '`1M伦敦同业拆借利率日元`')
```

```
iborpro_mysql('LIBOR-JPY-2M', '`2M伦敦同业拆借利率日元`')
iborpro_mysql('LIBOR-JPY-3M', '`3M伦敦同业拆借利率日元`')
iborpro_mysql('LIBOR-JPY-8M', '`8M伦敦同业拆借利率日元`')
iborpro_mysql('EURIBOR-EUR-1W', '`1W欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-2W', '`2W欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-3W', '`3W欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-1M', '`1M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-2M', '`2M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-3M', '`3M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-4M', '`4M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-5M', '`5M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-6M', '`6M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-7M', '`7M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-8M', '`8M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-9M', '`9M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-10M', '`10M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-11M', '`11M欧洲银行间同业拆借利率`')
iborpro_mysql('EURIBOR-EUR-1Y', '`1Y欧洲银行间同业拆借利率`')
iborpro_mysql('HIBOR-HKD-ON', '`ON香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-1W', '`1W香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-2W', '`2W香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-1M', '`1M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-2M', '`2M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-3M', '`3M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-4M', '`4M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-5M', '`5M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-6M', '`6M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-7M', '`7M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-8M', '`8M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-9M', '`9M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-10M', '`10M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-11M', '`11M香港同业拆借利率港元`')
iborpro_mysql('HIBOR-HKD-1Y', '`1Y香港同业拆借利率港元`')
iborpro_mysql('HIBOR-USD-ON', '`ON香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-1W', '`1W香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-2W', '`2W香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-1M', '`1M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-2M', '`2M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-3M', '`3M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-4M', '`4M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-5M', '`5M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-6M', '`6M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-7M', '`7M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-8M', '`8M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-9M', '`9M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-10M', '`10M香港同业拆借利率美元`')
iborpro_mysql('HIBOR-USD-11M', '`11M香港同业拆借利率美元`')
```

```
iborpro_mysql('HIBOR-USD-1Y', '`1Y香港同业拆借利率美元`')
iborpro_mysql('HIBOR-CNY-ON', '`ON香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-1W', '`1W香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-2W', '`2W香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-1M', '`1M香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-2M', '`2M香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-3M', '`3M香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-6M', '`6M香港同业拆借利率人民币`')
iborpro_mysql('HIBOR-CNY-1Y', '`1Y香港同业拆借利率人民币`')
iborpro_mysql('SIBOR-SGD-1M', '`1M新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-SGD-2M', '`2M新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-SGD-3M', '`3M新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-SGD-6M', '`6M新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-SGD-9M', '`9M新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-SGD-1Y', '`1Y新加坡同业拆借利率新加坡元`')
iborpro_mysql('SIBOR-USD-1M', '`1M新加坡同业拆借利率美元`')
iborpro_mysql('SIBOR-USD-2M', '`2M新加坡同业拆借利率美元`')
iborpro_mysql('SIBOR-USD-3M', '`3M新加坡同业拆借利率美元`')
iborpro_mysql('SIBOR-USD-6M', '`6M新加坡同业拆借利率美元`')
iborpro_mysql('SIBOR-USD-9M', '`9M新加坡同业拆借利率美元`')
iborpro_mysql('SIBOR-USD-1Y', '`1Y新加坡同业拆借利率美元`')
```

满满的满足感啊!



第 18 章 Stata 连接 MySQL 数据库的方案

Stata 可以通过 ODBC 连接数据库的，但是我试了很久也没有成功，然而昨天我想到了一个非常巧妙的方案······

昨天我成功的把 R 连接上了 MySQL 数据库，前天我介绍了 `rcall` 命令实现在 Stata 中运行 R 命令。于是我就想到了用 Stata 操作 R 连接数据库把数据读取再保存为 csv 文件，最后再由 Stata 读取。下面的代码就可以实现这个过程：

```
rcall: library(RMySQL); setwd("~/Desktop");engine = dbConnect(RMySQL::MySQL()
    , dbname = '交易数据', username = 'root', password = '12345', host = '
    localhost', port = 3306); ir = dbReadTable(engine, "002337前复权数据");
    write.table(ir, file = "test.csv", sep = ',')
import delimited test.csv, clear
```

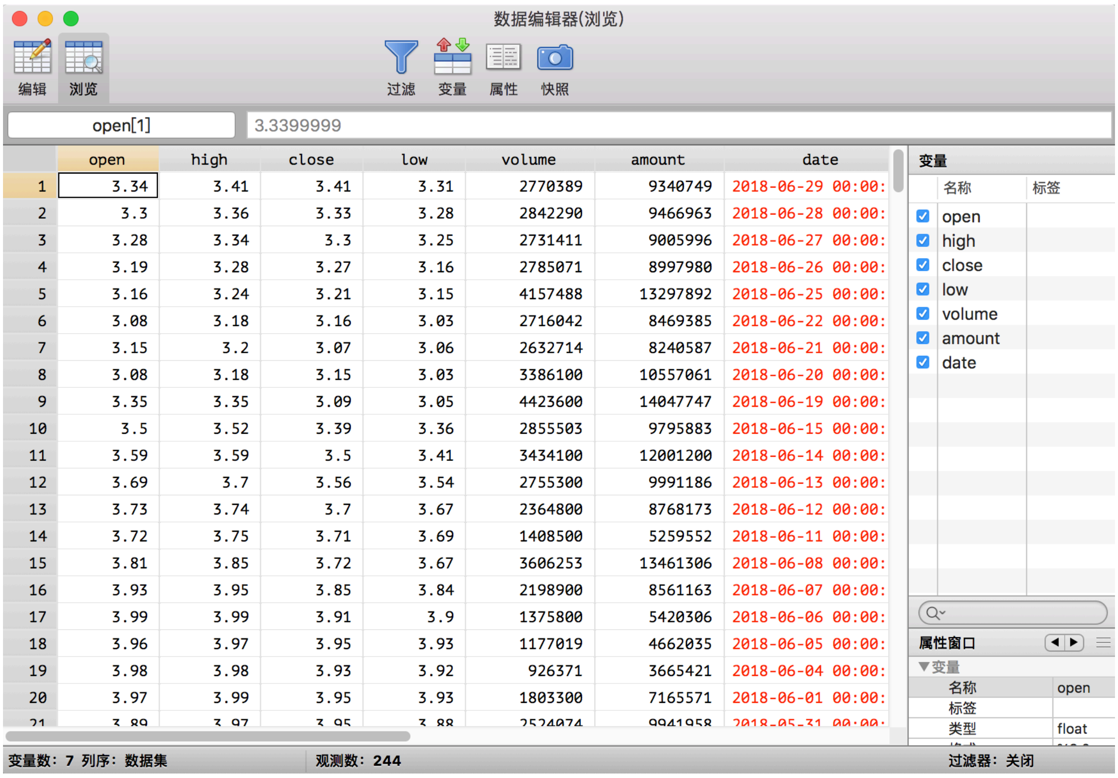
rcall 运行一大串 R 命令的时候只需要用分号隔开即可。为了方便读取数据我还写了下面的一个小命令：

```
cap prog drop mysql
prog def mysql
    version 14.0
    syntax, d(string) t(string)
    rcall: library(RMySQL); engine = dbConnect(RMySQL::MySQL(), dbname = '`d'
        ', username = 'root', password = '12345', host = 'localhost', port =
        3306); dbSendQuery(engine,'SET NAMES utf8'); ir = dbReadTable(engine,
         "`t'"); write.table(ir, file = "~/Desktop/test.csv", sep = ',', row.
        name = F)
    qui import delimited test.csv, encoding(utf8) clear varn(1)
    erase test.csv
end
```

这个命令有两个选项：- d()：选择数据库；- t()：选择数据库中的表格。

示例：

```
mysql, d(交易数据) t(002337前复权数据)
mysql, d(宏观经济数据) t(季度GDP)
mysql, d(投资参考数据) t(2018年分配预案)
mysql, d(新闻事件数据) t(000001相关新闻)
mysql, d(拆借利率) t(ON香港同业拆借利率美元)
```



	open	high	close	low	volume	amount	date
1	3.34	3.41	3.41	3.31	2770389	9340749	2018-06-29 00:00:
2	3.3	3.36	3.33	3.28	2842290	9466963	2018-06-28 00:00:
3	3.28	3.34	3.3	3.25	2731411	9005996	2018-06-27 00:00:
4	3.19	3.28	3.27	3.16	2785071	8997980	2018-06-26 00:00:
5	3.16	3.24	3.21	3.15	4157488	13297892	2018-06-25 00:00:
6	3.08	3.18	3.16	3.03	2716042	8469385	2018-06-22 00:00:
7	3.15	3.2	3.07	3.06	2632714	8240587	2018-06-21 00:00:
8	3.08	3.18	3.15	3.03	3386100	10557061	2018-06-20 00:00:
9	3.35	3.35	3.09	3.05	4423600	14047747	2018-06-19 00:00:
10	3.5	3.52	3.39	3.36	2855503	9795883	2018-06-15 00:00:
11	3.59	3.59	3.5	3.41	3434100	12001200	2018-06-14 00:00:
12	3.69	3.7	3.56	3.54	2755300	9991186	2018-06-13 00:00:
13	3.73	3.74	3.7	3.67	2364800	8768173	2018-06-12 00:00:
14	3.72	3.75	3.71	3.69	1408500	5259552	2018-06-11 00:00:
15	3.81	3.85	3.72	3.67	3606253	13461306	2018-06-08 00:00:
16	3.93	3.95	3.85	3.84	2198900	8561163	2018-06-07 00:00:
17	3.99	3.99	3.91	3.9	1375800	5420306	2018-06-06 00:00:
18	3.96	3.97	3.95	3.93	1177019	4662035	2018-06-05 00:00:
19	3.98	3.98	3.93	3.92	926371	3665421	2018-06-04 00:00:
20	3.97	3.99	3.95	3.93	1803300	7165571	2018-06-01 00:00:
21	3.89	3.97	3.95	3.88	2524074	9941958	2018-05-31 00:00:

你别说，还真的挺好用！不过这样还是很难实现将数据写入数据库的操作。

第 19 章 如何解决 MySQL 导出数据时的权限设置问题?

昨天在从 MySQL 中导出备份数据的时候遇到下面这样的错误:

```
# 备份数据库db_school.tb_student
select * from db_school.tb_student
into outfile '/Users/mr.cheng/Desktop/backupfile.txt'
fields terminated by ','
optionally enclosed by '"'
lines terminated by '?';
-- 结果:
[2018-09-05 10:40:44] [HY000][1290] The MySQL server is running with the --
    secure-file-priv option so it cannot execute this statement
```

搜索了很多解决办法,最后大概明白了是没有权限导入导出数据。,昨天搞到了一点终于弄好了。

首先可以使用下面的语句显示 secure-file-priv:

```
show variables like '%secure%';
-- 结果
require_secure_transport    OFF
secure_file_priv    NULL
```

NULL 就表示没有任何导入导出权限,所以需要修改这个参数:在`/private/ect/`里新建一个 `my.cnf` 文件,里面输入下面的内容

```
[mysqld]
secure_file_priv = /Users/mr.cheng/sql
```

`/Users/mr.cheng/sql` 是要授予导入导入导出权限的文件夹。似乎必须是用户下面的顶级文件夹,要不然会出错。

完成了这一步之后,重启 MySQL(从系统偏好那里重启)。然后再次运行下面的导出语句:

```
select * from db_school.tb_student
into outfile '/Users/mr.cheng/sql/backupfile.txt'
fields terminated by ','
optionally enclosed by '"'
lines terminated by '?';
```

结果仍然会出错。这个时候的提示是 `Permission denied`。这是因为 MySQL 对刚刚那个文件夹没有权限,所以下面一个让我非常头大的事情。。。我得把我在那个文件夹的权限转让给 MySQL,在终端输入下面的语句就能完成:

```
sudo chown mysql:mysql ~/sql/
```

转让权限之后。。。你就失去了对这个文件夹的直接控制权，收回权限可以用下面的语句：

```
sudo chown mr.cheng ~/sql/
```

然后就能把备份文件导出到这个文件夹了：

```
select * from db_school.tb_student
into outfile '/Users/mr.cheng/sql/backupfile.txt'
fields terminated by ','
optionally enclosed by '"'
lines terminated by '?';
```

第 20 章 PHP7.2 连接 MySQL8.0 失败的问题解决

似乎每天晚上都会遇到一个让人难以入眠的问题。昨天遇到的是 PHP 连接 MySQL 数据库的问题。不过昨天晚上没能解决,今天上午弄了一上午终于找到了这篇文章:php7 连接 mysql8。

最开始是运行下面的代码:

```
<?php
$con = mysqli_connect("localhost", "root", "12345");
if(!$con)  {
    echo "连接失败! <br>";
    echo "错误编号:".mysqli_errno()."<br>";
    echo "错误信息:".mysqli_error()."<br>";
}
else{
    echo "连接成功";
}
```

结果总是提示出错,记不清是什么样的错误了。大概解决方法如下。

按照上面提到的那篇博客中所说的,出错的原因在于:**mysql8 使用 caching_sha2_password 的身份验证机制,而以往的验证机制则是 mysql_native_password**。

首先需要打开 my.cnf,在 [mysqld] 标签下添加下面的配置:

```
default_authentication_plugin=mysql_native_password
```

重启 MySQL 之后会发现问题还是没有被解决。因为这样只对修改配置并重启 Mysql 后的新建用户有效,而最初创建的 root 用户依然是 caching_sha2_password 的身份验证机制。

下面继续修改配置文件 my.cnf,加上 (也是在哪个标签下):

```
skip-grant-tables
```

重启 MySQL 之后再删除刚刚添加的 `skip-grant-tables`,然后再次重启 MySQL。

在这之后,我就神奇的发现我的 PHP 能够连接 MySQL 了!(最后几步确实很迷。)

第 21 章 彻底解决了 Stata 与 MySQL 数据交换的问题

昨天晚上睡前就在思考这个问题，之前那个 Stata 调用 R 再调用 MySQL 的方法并不是非常好用。最近看了关于 MySQL 备份和恢复的部分，知道了 MySQL 可以通过一些命令行工具进行备份和恢复。于是就诞生了三个 Stata 命令，分别用于解决 Stata 读取、存储和传递 MySQL 代码的问题。

21.1 MySQL 实用工具

MySQL 提供了很多实用的命令行工具，这里我实用了 3 个。第一个是 mysqldump，用于导出数据库中的数据到文件中；第二个是 mysqlimport，用于把文本文件导入数据库中；第三个是 mysql 可以直接在终端执行某个 sql 文件。

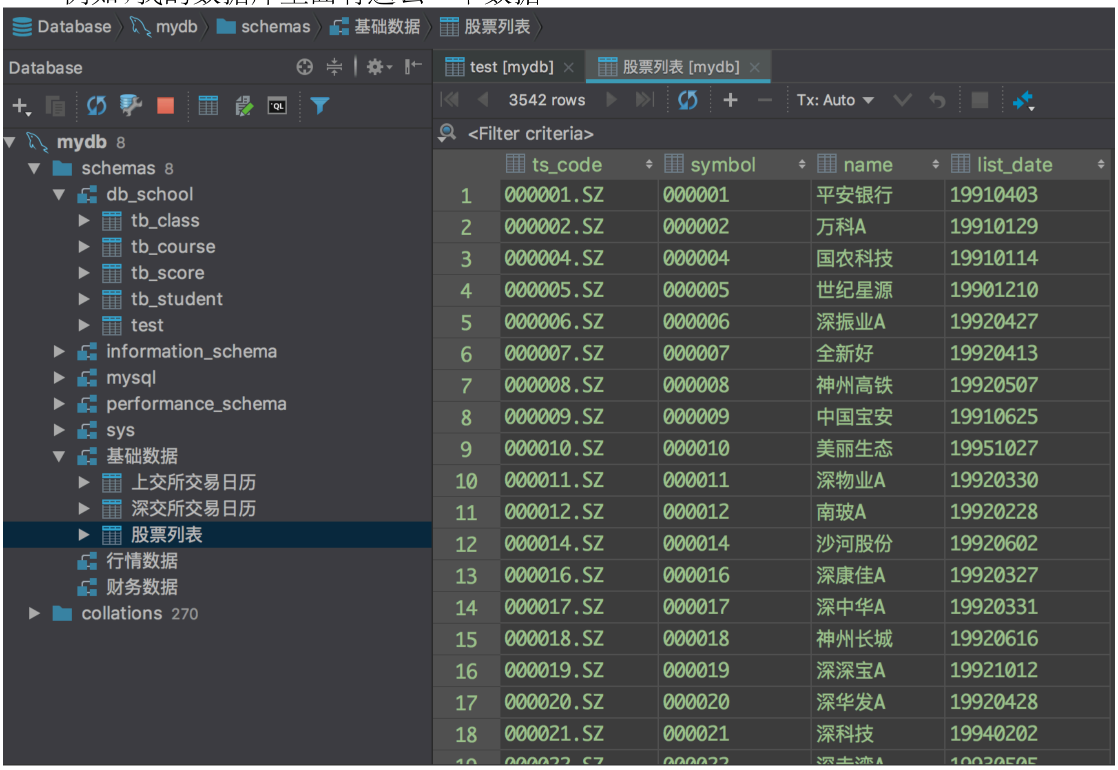
21.2 在 Stata 中读取 MySQL 中的数据

这部分用到的是 mysqldump 命令，思路就是先调用这个命令把需要读取的数据导出为 sql 文件（因为直接读出为 txt 文件在我的电脑上会遇到权限问题，就很烦人），然后简单处理一下再读入 Stata 即可，我把这个过程封装为一个命令：mysqli.ado，具体代码如下（需要注意这个代码需要结合你的电脑的实际情况进行修改）：

```
*! mysqli: 在Stata中读取MySQL中的数据
*! 程振兴 2018年9月8日
*! 【用法】
*!     mysqli database_name table_name[, varname(name1, name2, ··· , namen)]
*! database_name: 数据库的名称
*! table_name:    数据表的名称
*! varname:       可选项, 用于指定读取Stata后的变量名称
*! 【例如】
*!     mysqli db_school tb_student, v(学号 姓名 性别 出生日期 籍贯 民族 班级
    )
*!     mysqli 基础数据 股票列表, v(交易代码 股票代码 公司名称 挂牌日期)
*!     mysqli 基础数据 上交所交易日历, v(交易所 交易日期 是否开盘)
cap prog drop mysqli
prog def mysqli
version 14.0
syntax anything[, Varname(string)]
    qui{
        !/usr/local/mysql/bin/mysqldump -u root -p12345 `anything' | tr "(" "
           \n" > temp.txt
        infix strL v 1-20000 using temp.txt, clear
        keep if index(v, "),") | index(v, ");")
```

```
        drop if index(v, "\`")
        replace v = subinstr(v, `"'"', "", .)
        replace v = subinstr(v, `"),"', "", .)
        replace v = subinstr(v, `");"', "", .)
        split v, parse(,)
        loca varnum = r(nvars)
        drop v
        if "`varname'" != ""{
            tokenize `varname'
            forval i = 1/`varnum'{
                ren v`i' ``i''
            }
        }
        erase temp.txt
    }
end
```

例如,我的数据库里面有这么一个数据:



运行下面的 Stata 命令就可以把它读入 Stata 中:

```
mysqli 基础数据 股票列表, v(交易代码 股票代码 公司名称 挂牌日期)
```

	交易代码	股票代码	公司名称	挂牌日期
1	000001.SZ	000001	平安银行	19910403
2	000002.SZ	000002	万科A	19910129
3	000004.SZ	000004	国农科技	19910114
4	000005.SZ	000005	世纪星源	19901210
5	000006.SZ	000006	深振业A	19920427
6	000007.SZ	000007	全新好	19920413
7	000008.SZ	000008	神州高铁	19920507
8	000009.SZ	000009	中国宝安	19910625
9	000010.SZ	000010	美丽生态	19951027
10	000011.SZ	000011	深物业A	19920330
11	000012.SZ	000012	南玻A	19920228
12	000014.SZ	000014	沙河股份	19920602
13	000016.SZ	000016	深康佳A	19920327
14	000017.SZ	000017	深中华A	19920331
15	000018.SZ	000018	神州长城	19920616
16	000019.SZ	000019	深深宝A	19921012
17	000020.SZ	000020	深华发A	19920428
18	000021.SZ	000021	深科技	19940202
19	000022.SZ	000022	深赤湾A	19930505
20	000023.SZ	000023	深天地A	19930429

21.3 把 Stata 的数据存入 MySQL 数据库

这里用到了 mysqlimport 命令，思路是将 Stata 数据导出为 txt 文件再使用这个命令读入数据库即可。需要注意，要把数据中的空值替换成\N。同时需要在终端运行下面的命令收回 mysql 对 sql 文件夹的（在前面的文章中有说，这个文件夹是我授予 mysql 存储权限的文件夹）的权限，要不然 Stata 无法将数据存入这个文件夹（即使收回权限，mysqlimport 也是可以将数据存入数据库的）：

```
sudo chown mr.cheng ~/sql
```

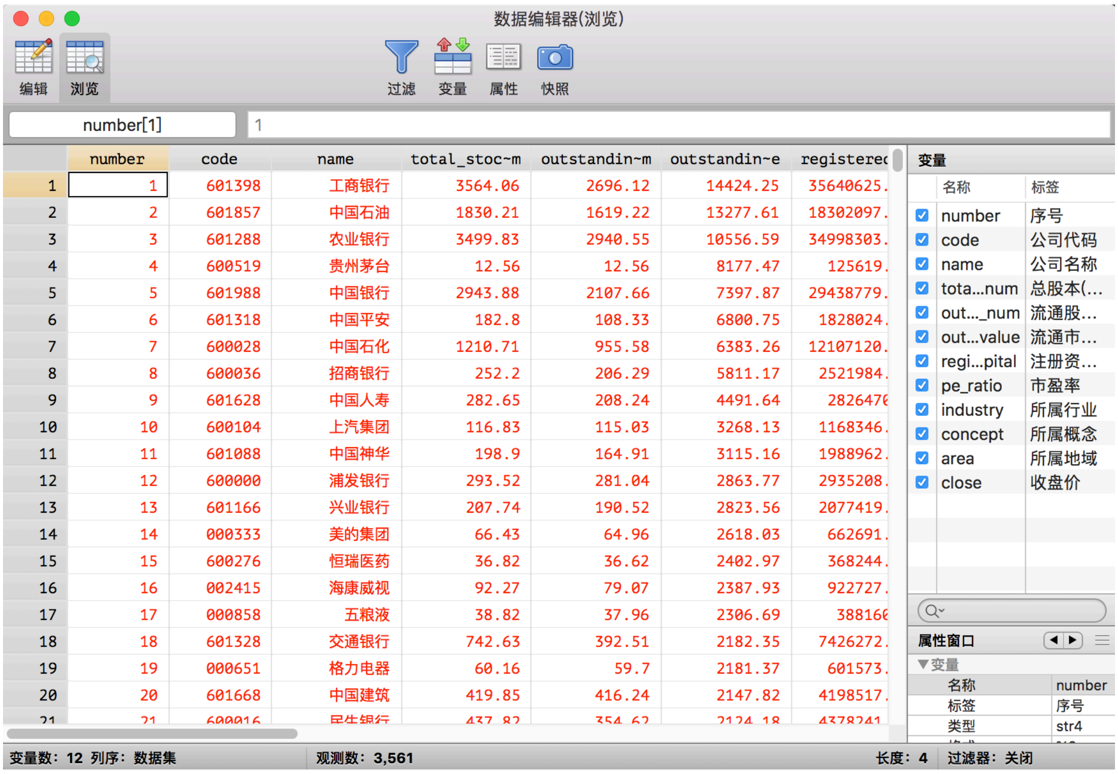
然后我把这部分代码封装成了一个 `mysqlj.ado`。这个名字的来源是因为那个叫 `mysqli.ado`。。。为什么那个要叫 `mysqli.ado` 呢，因为之前的那个 Stata 调用 R 再调用 MySQL 封装的叫 `mysql.ado`，所以这里想 `improved` 一下。

```
*! mysqlj: 将Stata中的数据存入MySQL中
*! 程振兴 2018年9月8日
*! 【用法】
```

```
*!      mysqli database_name table_name
*!  database_name:   数据库的名称
*!  table_name:      数据表的名称
*!  【注意】: 需要提前数据库中建立好对应的表格, 而且数据库和表的名称中不能含有
    中文!
*!  【例如】
*!      cnstock2
*!      mysqlj db_school test
cap prog drop mysqlj
prog def mysqlj
    version 14.0
    syntax anything
    qui{
        tokenize `anything'
        tostring _all, replace
        foreach i of varlist _all{
            cap replace `i' = "\N" if `i' == "."
        }
        export delimited using "/Users/mr.cheng/sql/`2'.txt", delimiter(tab)
            novarnames replace
        !/usr/local/mysql/bin/mysqlimport -u root -p12345 --replace `1' /
            Users/mr.cheng/sql/`2'.txt
        erase /Users/mr.cheng/sql/`2'.txt
    }
end
```

例如,我用 cnstock2 命令下载一份股票数据:

```
cnstock2
```



运行下面的命令即可存入数据库:

```
mysqlj db_school test
```

需要注意,这里不能使用中文作为数据库的名称和表的名称,因为会产生乱码而导致找不到数据库。而且这个表必须要提前创建好。例如,这里的这个表的创建代码为:

```
drop table if exists db_school.test;
create table db_school.test
(
  number int,
  code char(6) primary key,
  name varchar(15),
  total_stock_num float,
  outstanding_stock_num float,
  outstanding_stock_value float,
  registered_capital float,
  pe_ratio float,
  industry varchar(40),
  concept varchar(20),
  area varchar(20),
  close float
)engine = InnoDB;
```

21.4 在 Stata 中执行 MySQL 代码

这一部分和之前的 pycall 一样了，就是先写个 sql 文件，然后用 mysql 执行。我把这个过程封装成了一个命令——`mysqlcall.ado`：

```
*! mysqlcall: 在Stata中运行MySQL代码
*! 程振兴 2018年9月8日
*! 该命令的实现原理是将MySQL代码保存为一个临时的sql文件，
*! 然后将该文件使用命令行的mysql运行，然后将结果返回。
*! 一段示例的原始代码如下：
*! ###################### !*
*! ###       原理      ### !*
*! ###################### !*
*! file open myfile using temp.sql, write replace
*! file write myfile "drop table if exists db_school.test;" _n
*! file write myfile "create table db_school.test" _n
*! file write myfile "(" _n
*! file write myfile "  number int," _n
*! file write myfile "  code char(6) primary key," _n
*! file write myfile "  name varchar(15)," _n
*! file write myfile "  total_stock_num float," _n
*! file write myfile "  outstanding_stock_num float," _n
*! file write myfile "  outstanding_stock_value float," _n
*! file write myfile "  registered_capital float," _n
*! file write myfile "  pe_ratio float," _n
*! file write myfile "  industry varchar(40)," _n
*! file write myfile "  concept varchar(20)," _n
*! file write myfile "  area varchar(20)," _n
*! file write myfile "  close float" _n
*! file write myfile ")engine = InnoDB;" _n
*! file close myfile
*! !/usr/local/mysql/bin/mysql -u root -p12345 db_school < temp.sql
*! ###################### !*
*! ###       示例      ### !*
*! ###################### !*
*! mysqlcall, s
*! mysqlcall "drop table if exists db_school.test;"
*! mysqlcall "create table db_school.test"
*! mysqlcall "("
*! mysqlcall "  number int,"
*! mysqlcall "  code char(6) primary key,"
*! mysqlcall "  name varchar(15),"
*! mysqlcall "  total_stock_num float,"
*! mysqlcall "  outstanding_stock_num float,"
*! mysqlcall "  outstanding_stock_value float,"
*! mysqlcall "  registered_capital float,"
*! mysqlcall "  pe_ratio float,"
```

```
*! mysqlcall "  industry varchar(40),"
*! mysqlcall "  concept varchar(20),"
*! mysqlcall "  area varchar(20),"
*! mysqlcall "  close float"
*! mysqlcall ")engine = InnoDB;"
*! mysqlcall, e r
cap prog drop mysqlcall
prog def mysqlcall
    version 14.0
    syntax [anything(name = sqlcode)] [, Start End Run]
    if "`start'" != ""{
        cap file close myfile
        file open myfile using tempsqlfile.sql, write replace
    }
    if "`end'" != ""{
        file close myfile
    }
    else{
        file write myfile `sqlcode' _n
    }
    if "`run'" != ""{
        !/usr/local/mysql/bin/mysql -u root -p12345 < tempsqlfile.sql
        erase tempsqlfile.sql
    }
end
```

例如，通过下面的工作流程就能实现创建表格、存储数据了：

```
/* 创建表格 */
mysqlcall, s
mysqlcall "drop table if exists db_school.test;"
mysqlcall "create table db_school.test"
mysqlcall "("
mysqlcall "  number int,"
mysqlcall "  code char(6) primary key,"
mysqlcall "  name varchar(15),"
mysqlcall "  total_stock_num float,"
mysqlcall "  outstanding_stock_num float,"
mysqlcall "  outstanding_stock_value float,"
mysqlcall "  registered_capital float,"
mysqlcall "  pe_ratio float,"
mysqlcall "  industry varchar(40),"
mysqlcall "  concept varchar(20),"
mysqlcall "  area varchar(20),"
mysqlcall "  close float"
mysqlcall ")engine = InnoDB;"
mysqlcall, e r
```

```
/* 下载、存储数据 */
cnstock2
mysqlj db_school test
```

好了，我要把三个 do 文件放进我的系统文件夹了。

参考文献

DONG W, 2018. pymysql pymysql.err.OperationalError 1045 Access denied 最简单解决办法[EB/OL]. https://blog.csdn.net/dongweionly/article/details/80273095.

KUMU, 2017. 解决 Access denied for user 'root'@'localhost' (using password: NO)[EB/OL]. https://my.oschina.net/u/2294525/blog/820625.

MAIRR, 2018. Data Grip 使用—> 创建数据库连接[EB/OL]. https://www.cnblogs.com/Mairr/p/8406844.html.

钺览 IT, 2018. php7 连接 mysql8[EB/OL]. https://www.cnblogs.com/roggeyue/articles/9191884.html.