# Go语言数据库编程

本章重点为大家介绍如下的内容：

* 什么是数据库
* 关系型数据库
* 数据库范式
* MySQL的特点
* MySQL安装软件下载及安装（noinstall安装及图形化方式安装）
* 启动和停止MySQL服务
* 客户端管理工具Navicat
* MySQL数据类型
* SQL基础入门
* SQL中运算符及常用函数
* DQL语句高级查询
* 约束
* 数据库的其他内容(事务、视图、索引、触发器)
* Go程序中实现MySQL数据库操作

## 数据库介绍

### 什么是数据库

1、数据库(database)就是存储数据的仓库。

● 数据库（Database）是按照数据结构来组织、存储和管理数据的仓库，每个数据库都有不同的API用于创建、存储、管理数据。

● 为了方便数据的存储和管理，将数据按照特定的规律存储在磁盘上。通过数据库管理系统，有效地组织和管理存储在数据库中的数据。

● 数据库（database）是按照数据结构来组织、存储和管理数据的仓库（通常是一个文件或一组文件）。

● 理解数据库的一种最简单的办法就是将其想象为一个文件柜，此文件柜是一个存放数据的物理位置。

2、为什么要用数据库

● 数据其实可以存储在文本文件中，但是只适用于数据量很少的时候，数据量很少的情况下，在文件中保存数据、读写数据，操作简单便利且速度快；但是当数据量稍微大的情况，普通文本文件保存数据的方式就显得捉襟见肘，甚至束手无策。比如数据结构、数据类型无法设计，数据读写方式复杂且速度慢，对于复杂的数据查询更是无法做到。所以需要专业的数据库软件；

● 存储数据也可以使用微软的 Excel，但是，随着数据越来越多，在 Excel这样的电子表格里查找数据就变得很慢，而且不可靠。所以需要更高效的数据库管理软件。

3、数据库分类

● 而在当今的互联网中，最常见的数据库模型主要是两种，即关系型数据库和非关系型数据库。

● 关系型数据库包括：Oracle、MySQL、SQLServer、Access、DB2等

● 非关系型数据库包括：

○ 键值存储型数据库：Redis

○ 文档存储型数据库：MongoDB、CouchDB

○ 列表存储型数据库：HBase

○ 图数据库：FlockDB

4、数据库中几个名词

● DB：是指datebase（数据库）

数据库是存储数据的一个集合，数据库中通常使用数据表等组成，而数据表是由数据的字段和数据的值等信息组成。

● DBMS:是指datebase mangement systerm（数据库管理系统）

它是操作数据库和管理数据库的一个系统，比如mysql、sqlserver等都是属于数据库管理软件，人们通过这些系统或者工具来管理数据库内的数据。

● DBS:是指datebase systerm (数据库系统）

数据库系统又数据库和数据库管理软件等组成，数据库是一个逻辑上的存储数据的概念，而对应的是实体是数据库管理软件存储存储在硬盘上的数据库，所以数据库系统包含数据库和数据库管理软件。

### 关系型数据库

1、关系型数据库模型是把复杂的数据结构归结为简单的二元关系（即二维表格形式）。

在关系型数据库中，对数据的操作几乎全部建立在一个或多个关系表格上，通过对这些关联的表格分类、合并、连接或选取等运算来实现数据库的管理。

关系型数据库诞生40多年了，从理论产生发展到现实产品，例如：Oracle和MySQL，Oracle在数据库领域上升到霸主地位，形成每年高达数百亿美元的庞大产业市场。MySql数据库在中小企业和免费市场具有绝对地位。

2、关系型数据库中常用术语

1）、数据库表table:

● 表是数据库存储数据的基本单位。数据库正是一些互相关联的表的集合。

● 数据库中的表与Excel电子表格结构一致。

2）、列column:

● 列也叫做字段。将具有相同属性的数据放在同一列中。

● 例如用户名字段，该列中只包含每个会员的用户名账号信息。

3）、行row：

● 行也叫做记录。是一组相关的数据，例如用户账号表中每一条都保存的是一个用户的完整账号信息；

● 表中的数据是按行存储的，所保存的每个记录存储在自己的行内，如果将表想象为网格，网格中垂直的列为表列，水平行为表行。

4）、主键primary key：

● 每个表中数据绝对唯一的字段就是该表的主键。

● 一个数据表中只能包含一个主键。往往使用主键来标记每一条数据，从而查询数据。换句话说就是，表中每一行都应该有可以唯一标识自己的一列。

● 一个顾客表中可以使用顾客编码列，而订单表可以使用订单ID等。一个表中没有主键的话，更新或删除表中特定行很困难，因为没有安全的方法保证只涉及相关的行。

● 主键的规则表中的任何列都可以作为主键，只要它满足一下条件：

○ 任何两行都不具有相同的主键值（每一行的主键值是唯一的）。

○ 每个行都必须具有一个主键值（主键不允许null值）

● 主键的好习惯除了强制的规则外，应该坚持的几个普遍认可的好习惯：

○ 不更新主键列中的值

○ 不重用主键列的值

○ 不在主键列中使用可能会更改的值

3、关系型数据库中的关系

在关系型数据库中，弄清各个模块（或者叫实体或者叫表）之间的关系非常重要。关系型数据库中有三种基本关系：

● 1-1(一对一):比如一个国家对应一个首都

● 1-N(一对多):比如一个分类下可以有多个商品，一个班级有多个学生，这种关系往往存在从属关系

● N-N(多对多):比如学生和课程，一个学生可以选多门课，不同的学生也可以选择同一门课

### 数据库范式

1、范式（Normal Form）

● 冗余：不同数据库表中保存了完全相同的数据就产生了数据冗余。冗余降低了数据库的利用率甚至性能，但提高了数据的安全性和访问性；

● 为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。在实际开发中最为常见的设计范式有三个，即：第一范式（1NF），第二范式（2NF），第三范式（3NF）

2、第一范式(确保每列保持原子性)

● 第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

● 第一范式的合理遵循需要根据系统的实际需求来定。比如某些数据库系统中需要用到“地址”这个属性，本来直接将“地址”属性设计成一个数据库表的字段就行。但是如果系统经常会访问“地址”属性中的“城市”部分，那么就非要将“地址”这个属性重新拆分为省份、城市、详细地址等多个部分进行存储，这样在对地址中某一部分操作的时候将非常方便。这样设计才算满足了数据库的第一范式，如下表所示。



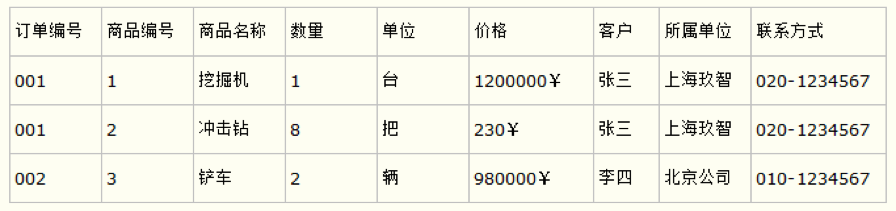
上表所示的用户信息遵循了第一范式的要求，这样在对用户使用城市进行分类的时候就非常方便，也提高了数据库的性能。

3、第二范式(确保表中的每列都和主键相关)

● 第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

● 比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键，如下表所示。

● 订单信息表



● 这样就产生一个问题：这个表中是以订单编号和商品编号作为联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。

● 而如果把这个订单信息表进行拆分，把商品信息分离到另一个表中，把订单项目表也分离到另一个表中，就非常完美了。如下所示。



● 这样设计，在很大程度上减小了数据库的冗余。如果要获取订单的商品信息，使用商品编号到商品信息表中查询即可。

4、第三范式(确保每列都和主键列直接相关,而不是间接相关)

● 第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

● 比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。如下面这两个表所示的设计就是一个满足第三范式的数据库表。



● 这样在查询订单信息的时候，就可以使用客户编号来引用客户信息表中的记录，也不必在订单信息表中多次输入客户信息的内容，减小了数据冗余。

## MySQL数据库介绍

### MySQL的正确发音

MySQL之父Monty给出了标准答案是：/mai es kjuː ɛl/ 或者 /mjuː es kjuː ɛl /，因为My芬兰语发音是/mjuː /。SQL在形成初期，也就是70年代， 曾经以SEQUEL （发音[ˈsi:kwəl]）（Structured English Query Language）作为其缩写，1980年才改名为SQL。所以如果非要读作/mai si:kwəl /也勉强可以接受。】

### MySQL的发展历史

● Michael Monty Widenius——MySQL之父（迈克尔·曼提·维德纽斯）

● Monty在1962年出生于芬兰的首都赫尔辛基（Helsinki）。（芬兰，对于中国人来说最著名的便是家喻户晓的诺基亚了）；

● Monty 1978年买了他的第一台个人电脑，为了买这台电脑他用了整个夏天在马路上铺沥青；

● 1980年，18岁的 Monty 沉迷于编程中，当其他人去参加聚会的时候，Monty 在家里写代码；

● Monty为了增加他电脑的存储空间，因为芬兰没有商店卖他想要的，后来他就乘船来到了瑞典的一家电脑店铺；

● 在瑞典Monty 见到了Allan Larsson、 David Axmark，他们三个保持联系了很多年，在项目上经常互相帮助，偶尔还一起合作。1995年他们一起创立了 MySQL AB（数据仓库公司）。Monty 负责编码，David 和 Allan 做剩下的事情；

● Monty为芬兰公司 Tapio Laakso OY （OY是芬兰 “有限公司”的缩写）工作的同时也为他和 Allan、David一起创建的数据仓库公司工作；

● 早在80年代，Monty用ISAM（Indexed Sequential Access Method，索引顺序存取方法）和mSQL（Mini SQL，一种轻量级SQL数据库引擎技术）来连接访问表格，后来发现mSQL的速度和灵活性不能满足需要，于是就着手开发几乎和mSQL API相同的数据库引擎。这就是MySQL的雏形；

● Monty遇到Allan 和 David 后，两位极力说服Monty继续完善之前的数据库代码并发布。 于是1996年10月，MySQL 终于发布了。MySQL 数据库表现很快、很可靠，学起来简单，管理维护容易；

● “My”是因为 Monty 的大女儿叫做 My（其实My是一个芬兰语名字，芬兰语发音是/mjuː /），而“SQL”就是 Structured Query Language（结构化查询语言）。所以MySQL的正确发音，Monty给出了标准答案是：/mai es kjuː ɛl/ 或者 /mjuː es kjuː ɛl /；

● 2005年，数据库巨头Oracle将开发InnoDB存储引擎的Innobase OY公司收入自己旗下；

● 在2008年1月16日MySQL AB被Sun公司收购，而2009年，SUN又被Oracle收购。

### MySQL的特点

● MySQL 是一个关系型数据库管理系统，目前属于 Oracle （甲骨文）旗下产品；

● MySQL 是最流行的关系型数据库管理系统之一。在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系型数据库管理系统) 应用软件。关系型数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性；

● MySQL所使用的 SQL 语言是用于访问数据库的最标准的结构化查询语言；

● MySQL支持大型的数据库。可以处理拥有上千万条记录的大型数据库。支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB，64位系统支持最大的表文件为8TB；

● MySQL支持运行于多个系统上，且支持多种语言。包括C、C++、Python、Java、Perl、PHP、Ruby等；

● 与其他的大型数据库，例如 Oracle、DB2、SQL Server等相比，MySQL 自有它的不足之处，但是这丝毫也没有减少它受欢迎的程度。对于一般的个人使用者和中小型企业来说，MySQL提供的功能已经绰绰有余。而且MySQL 采用了双授权政策，分为社区版和商业版，由于其体积小、速度快，尤其是开源这一特点，不需要支付额外的费用，总体拥有成本低，一般中小型网站的开发都选择 MySQL 作为网站数据库。

## 安装MySQL

### MySQL安装包下载

1、官方网站www.mysql.com下载最新版本的MySQL数据库

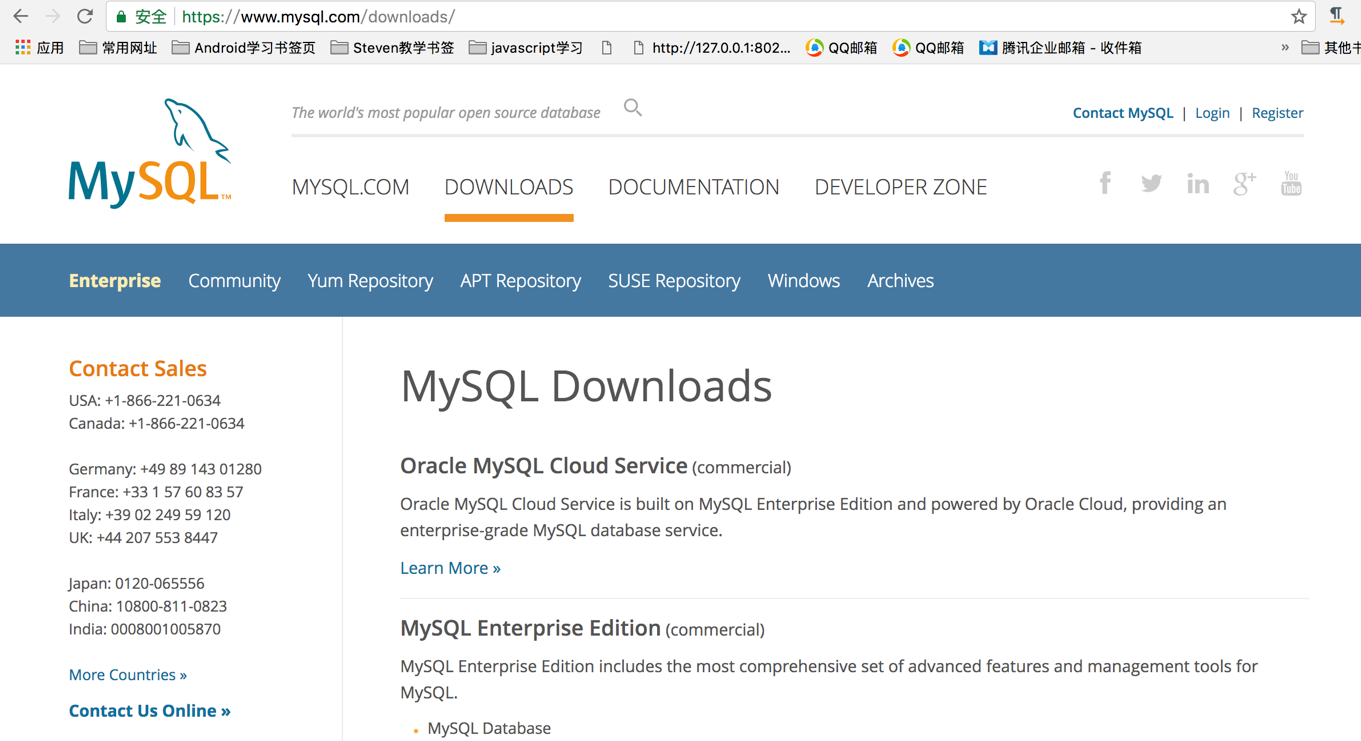


图 1

2、社区版（Community）和企业版（Enterprise）

● 目前MySQL数据库分为社区版（Community）和企业版（Enterprise）；

● 两者重要的区别：社区版是自由下载且完全免费的，但是官方不提供技术支持，适用于大多数普通用户；企业版是收费的，不能在线下载，当然它提供了更多的功能和更完备的技术支持，更适合于企业客户；

● 社区版的下载地址：https://dev.mysql.com/downloads

● 当前最新版本是MySQL Community Server 5.7.21



图 2

3、ZIP Archive与MSI Installer

● Windows系统下有两种安装包：ZIP Archive与MSI Installer；

● ZIP Archive：直接解压的免安装软件包；

● MSI是微软特有压缩格式，是需要安装的软件包；

● MySQL ZIP Archive： 虽然无需安装，但是关于参数配置、服务的启动关闭都只能通过手工在DOS命令窗口中实现。尤其是参数配置文件，比较复杂，对于初学者来说不容易理解；

● MySQL MSI Installer：虽然安装过程复杂，但安装引导提供了一个图形化的实例配置向导，引导用户逐步进行实例参数的设置。

（1）、ZIP Archive （直接解压的免安装软件包）

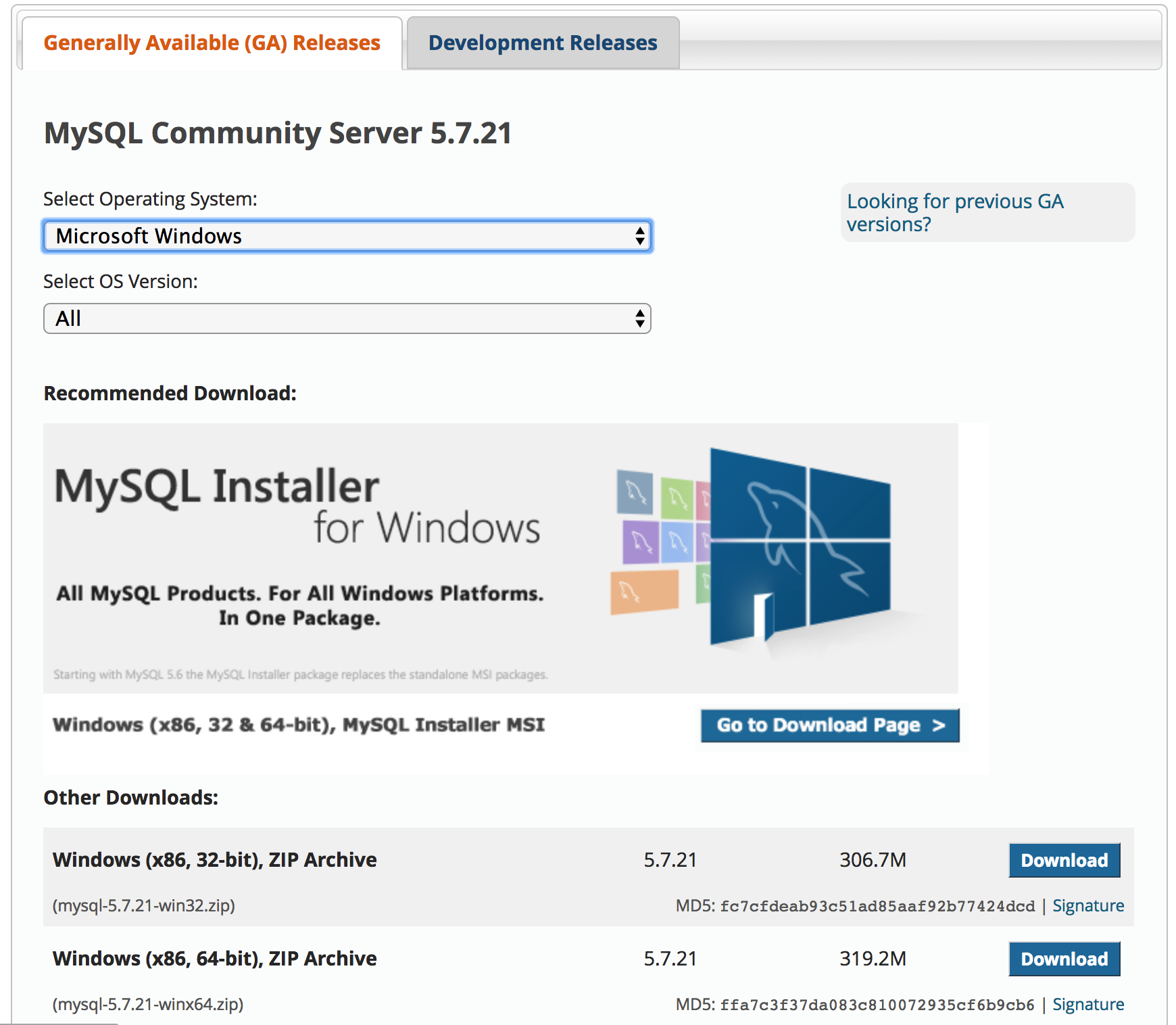


图 3

（2）、MSI Installer（微软特有压缩格式的安装软件包）

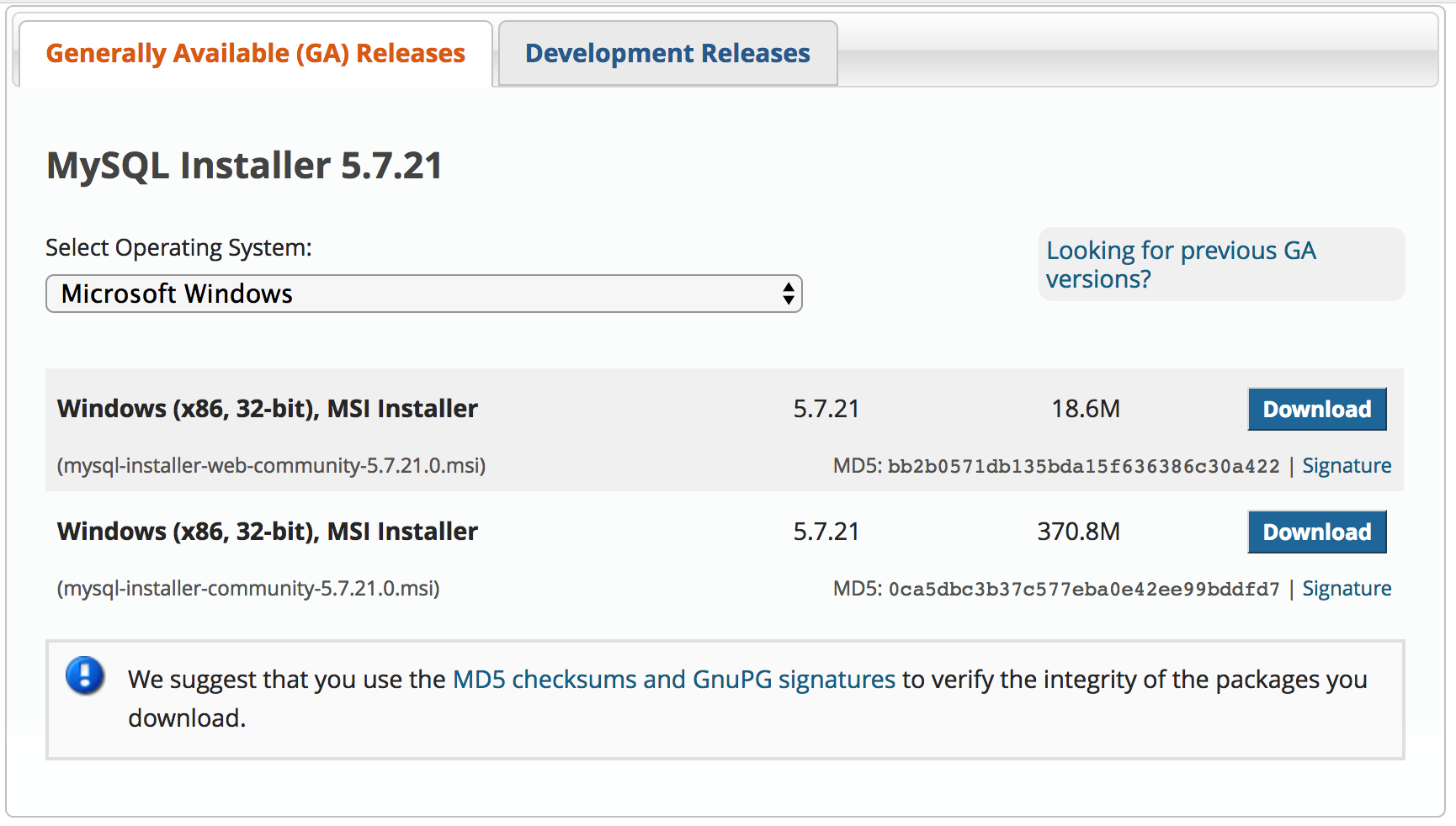


图 4

### MySQL软件noinstall安装

noinstall安装就是无需安装过程的MySQL安装方式。虽然无需安装，但是需要进行环境变量配置，手工写参数配置文件my.ini ，并将MySQL注册为windows服务。

1、下载安装文件，解压文件到指定目录;

比如解压到：C:\mysql-5.7.21-winx64

2、设置环境变量

操作如下：

1）右键单击我的电脑->属性->高级系统设置(高级)->环境变量

点击系统变量下的新建按钮

输入变量名：MYSQL\_HOME

输入变量值：C:\mysql-5.7.21-winx64

2）选择系统变量中的Path

点击编辑按钮

在变量值中添加变量值：%MYSQL\_HOME%\bin

注意是在原有变量值后面加上这个变量，用;隔开，不能删除原来的变量值

3、修改C:\mysql-5.7.21-winx64文件夹下的my.ini文件，如果没有则复制my-default.ini来进行修改：

#########################################################

[client]

port=3306

default-character-set=utf8

[mysqld]

port=3306

character\_set\_server=utf8

basedir=%MYSQL\_HOME%

#解压目录

datadir=%MYSQL\_HOME%\data

#解压目录下data目录

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

[WinMySQLAdmin]

%MYSQL\_HOME%\bin\mysqld.exe

########

4、将MySQL注册为windows系统服务

1）管理员身份运行cmd，从控制台进入到MySQL解压目录下的 bin 目录下；

2）输入服务安装命令；

mysqld install MySQL --defaults-file="%MYSQL\_HOME%\my.ini"

#解压目录下修改的my.ini文件

安装成功后会提示服务安装成功。

#注：my.ini文件放在MySQL解压后的根目录下

#移除服务命令为：mysqld -remove

### MySQL图形化方式安装

核心安装步骤

● 1、点击安装包，启动图形化安装界面，勾选“接受许可条款”；

● 2、选择安装类型。选择“开发者默认项（Developer Default）”；

● 3、检测必备条件；

● 4、等待安装，点击“Execute”会执行安装。安装完毕，会在每一项前显示绿色对勾；

● 5、产品配置；

● 6、类型及网络选择。选择默认项即可；

● 7、设置服务器配置类型。一般选择“开发者机器”，设置端口号为3306；

● 8、设置账号密码；

● 9、勾选“把MySQL设置成Windows服务”；

● 10、应用配置。配置完成会在前方显示绿色对勾；

● 11、打开计算机“服务”，启动和停止MySQL服务。

1、点击安装包mysql-installer-community-5.7.20.msi，启动图形化安装界面，勾选“接受许可条款”。

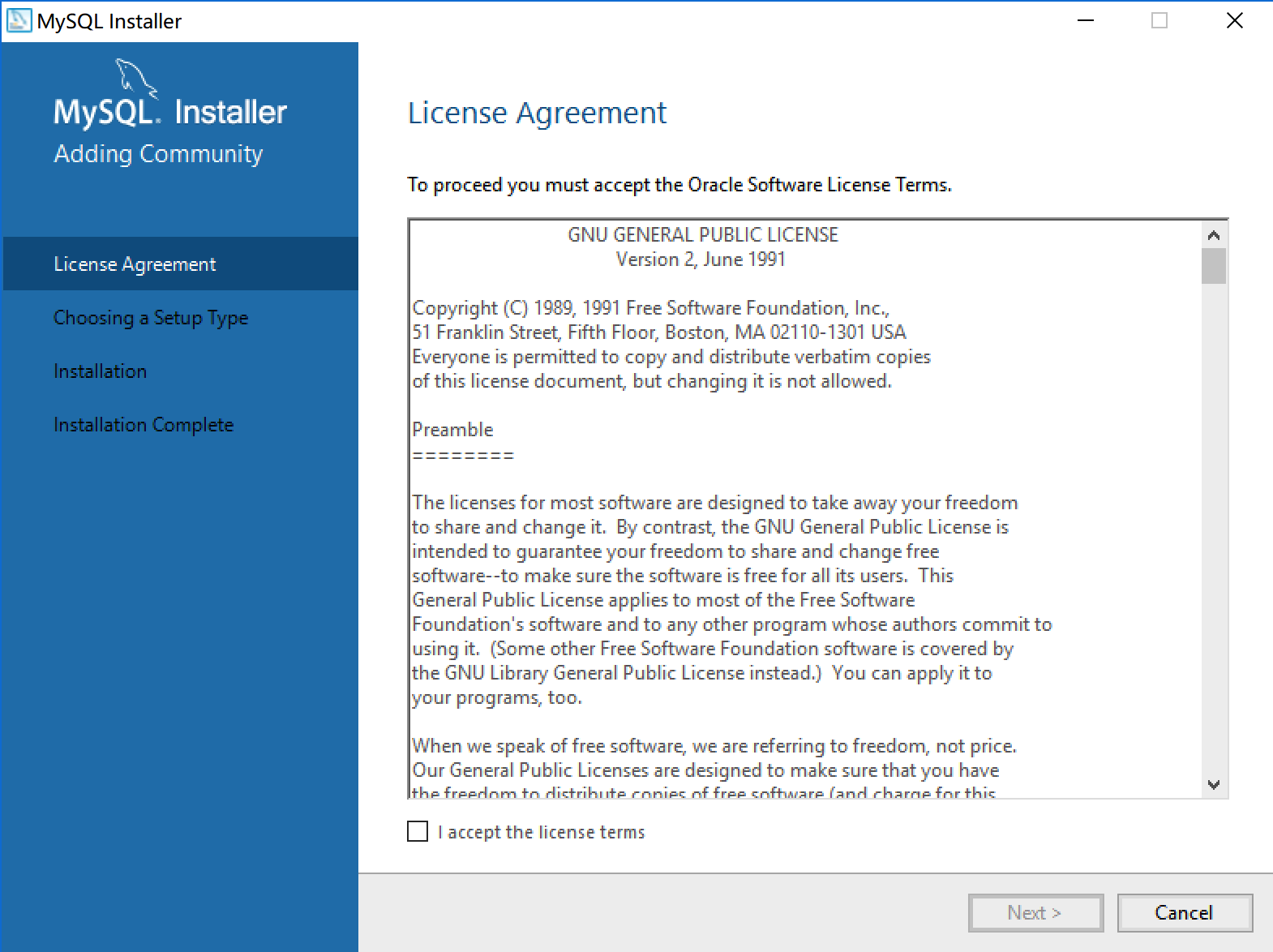


图 5

2、选择安装类型。选择“开发者默认项（Developer Default）”

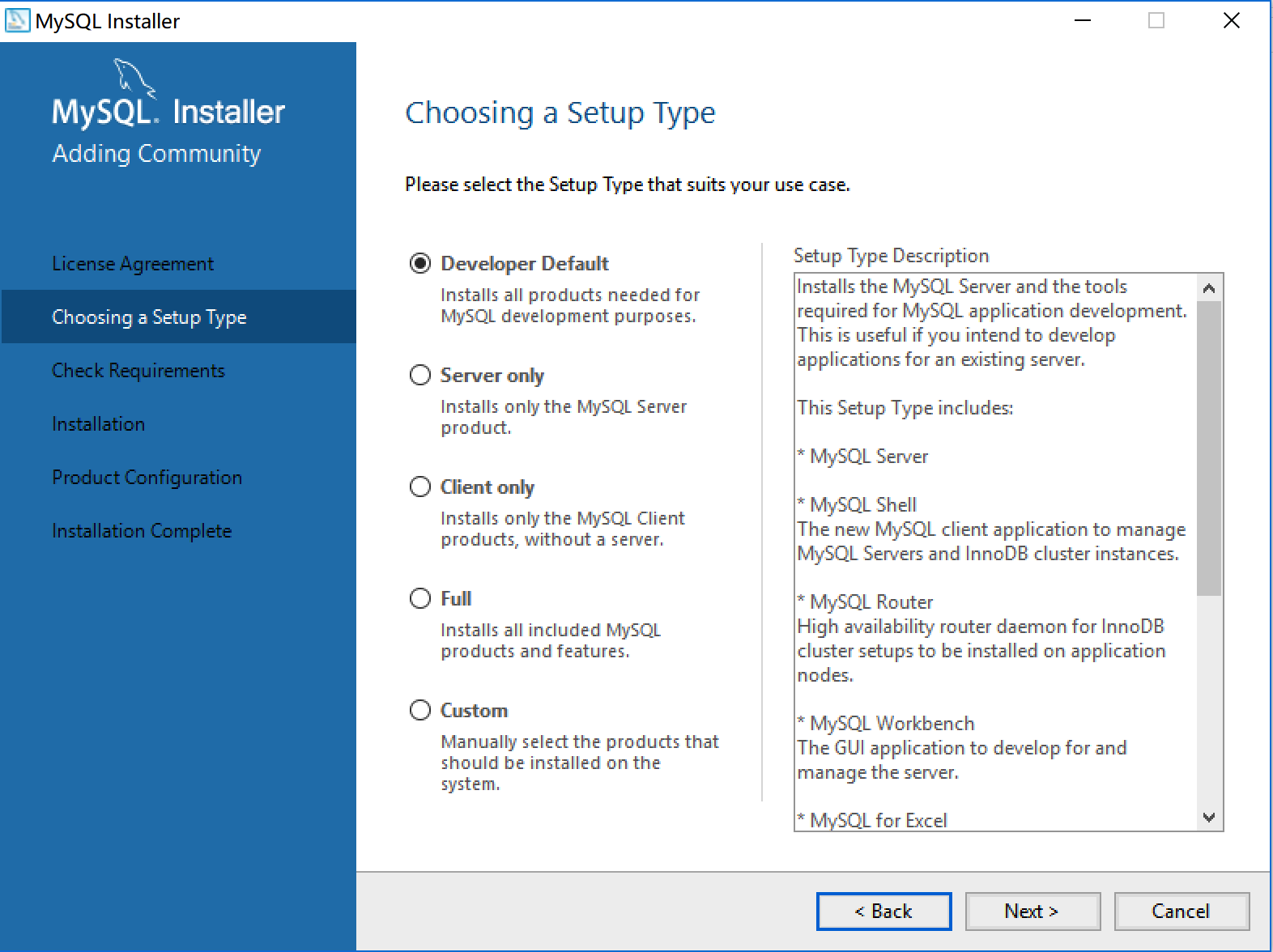


图 6

3、检测必备条件

● 方框中显示三列，第一列显示可以连接到MySQL的软件，比如Visual Studio，Eclipse，PyCharm等，中间列是需求的版本或者额外组件，第三列是状态；

● 选择一个选项，然后点击下面的check，如果有弹窗说明该软件没有安装需求的版本或者额外组件，如果已经安装了，则前面会多一个绿色的勾。说明可以使用；

● 如果没有达到要求，需要手动安装额外的软件；

● 如果有些产品你不需要用的话，不需要安装额外软件，直接点next就可以了。会弹出一个窗口，忽略他，直接点击“是”就可以了。

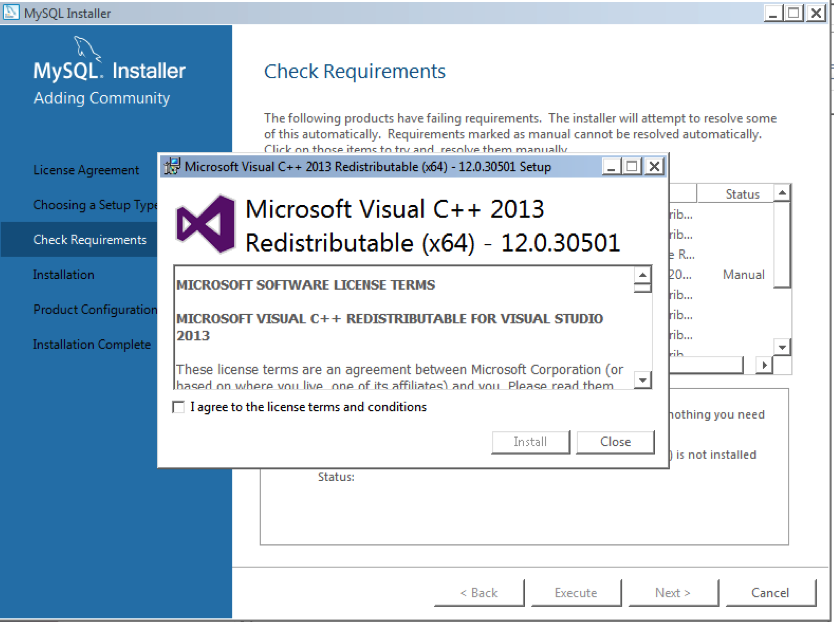


图 7

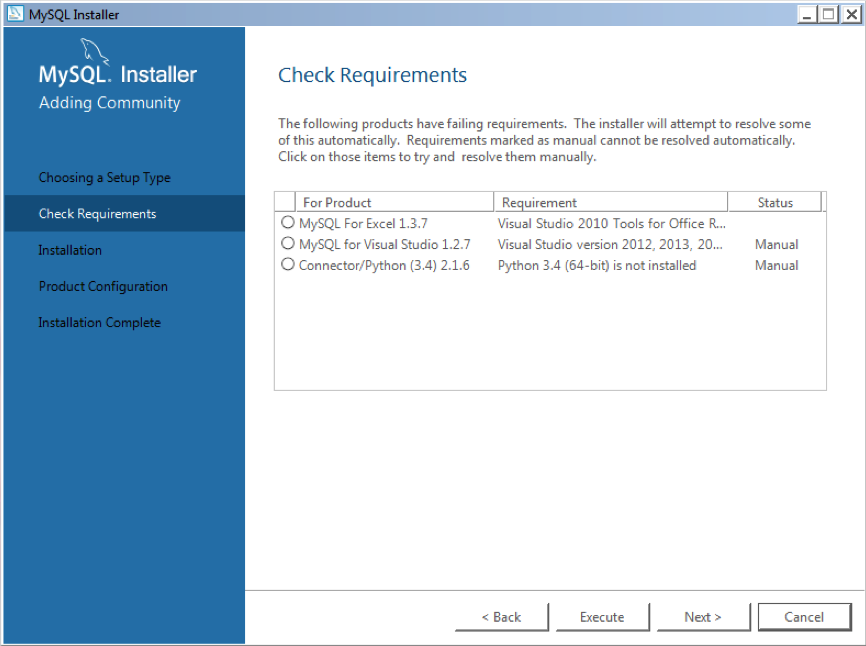


图 8

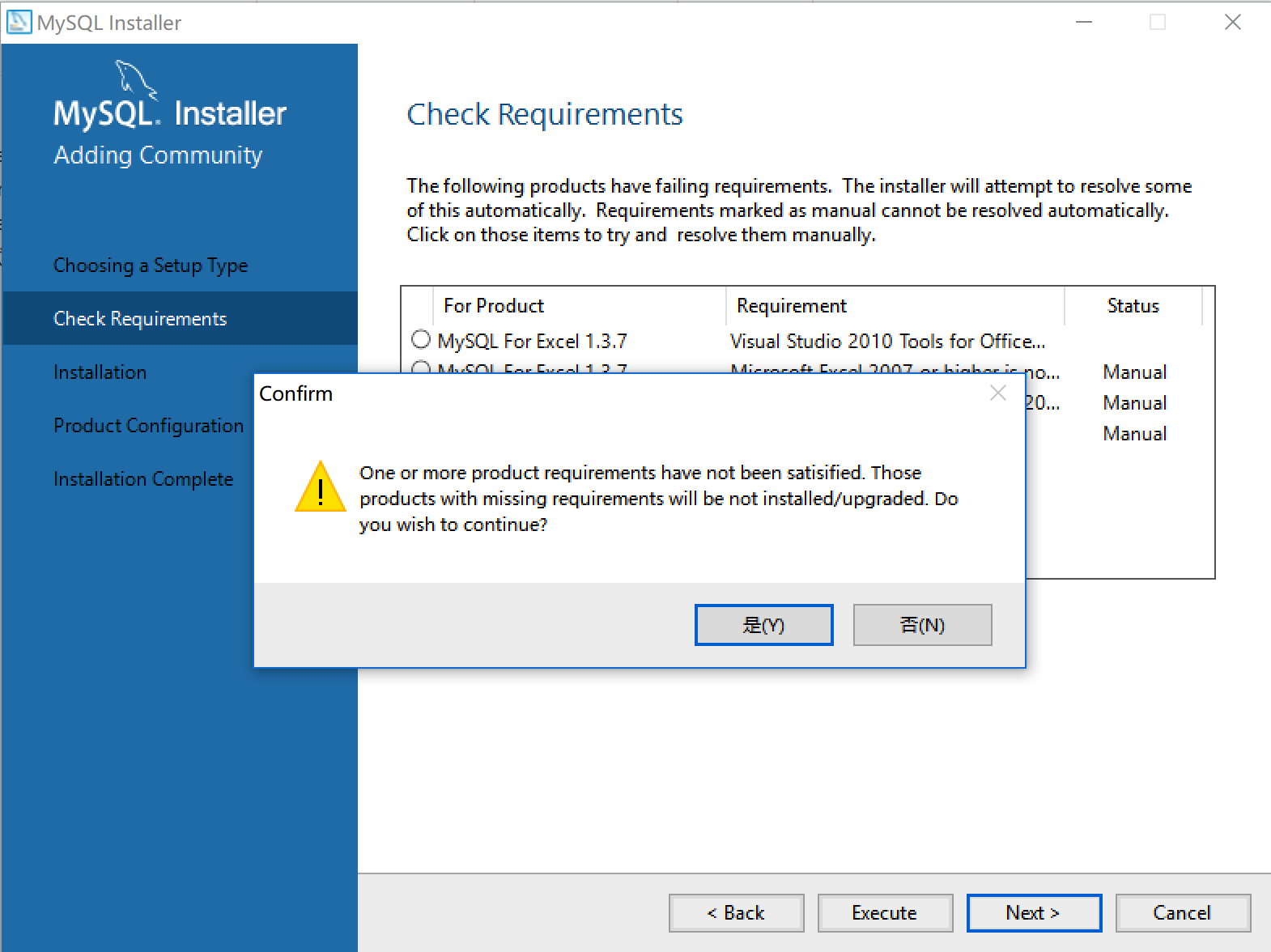


图 9

4、等待安装，点击“Execute”会执行安装。安装完毕，会在每一项前显示绿色对勾✅。

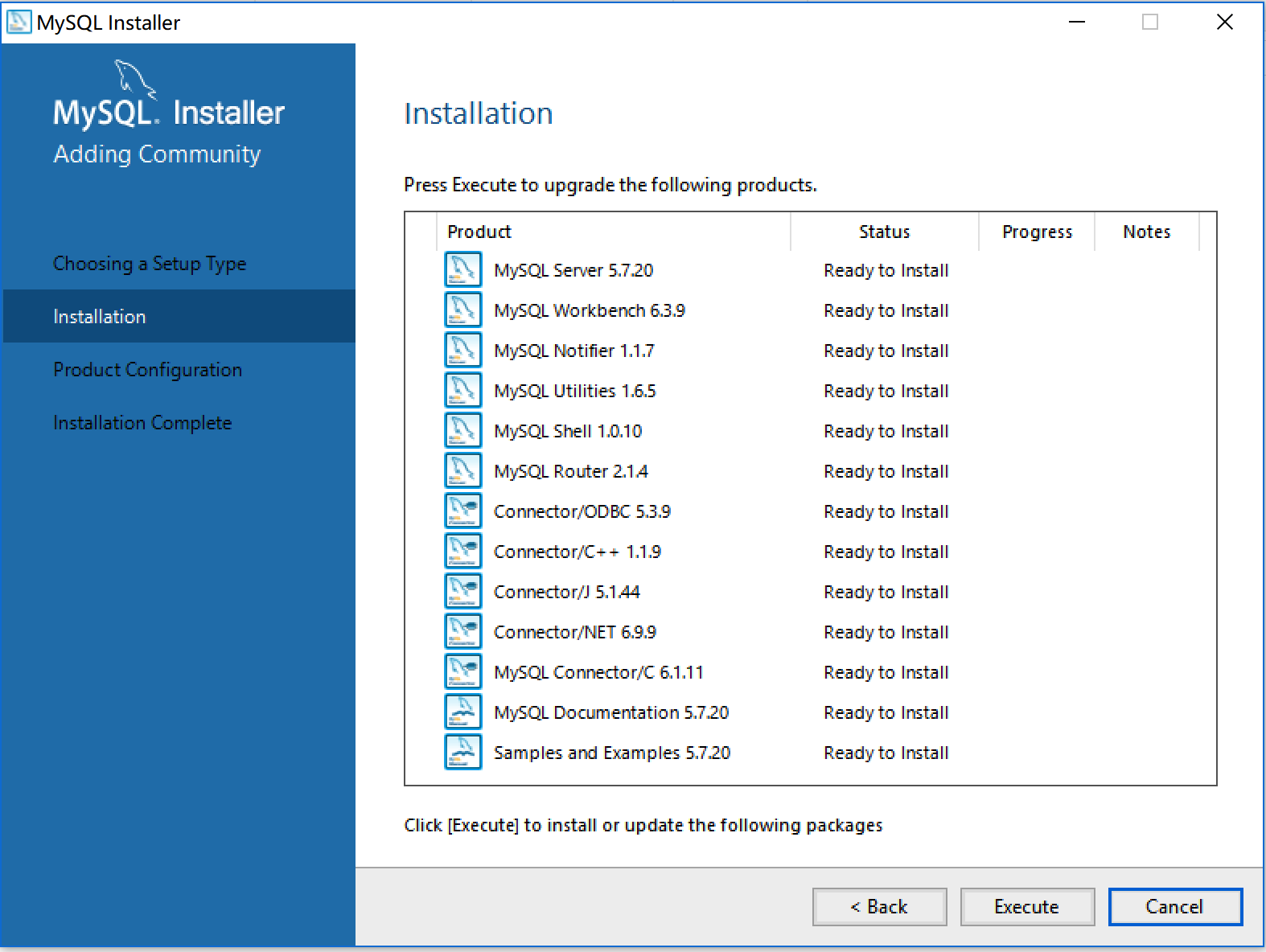


图 10

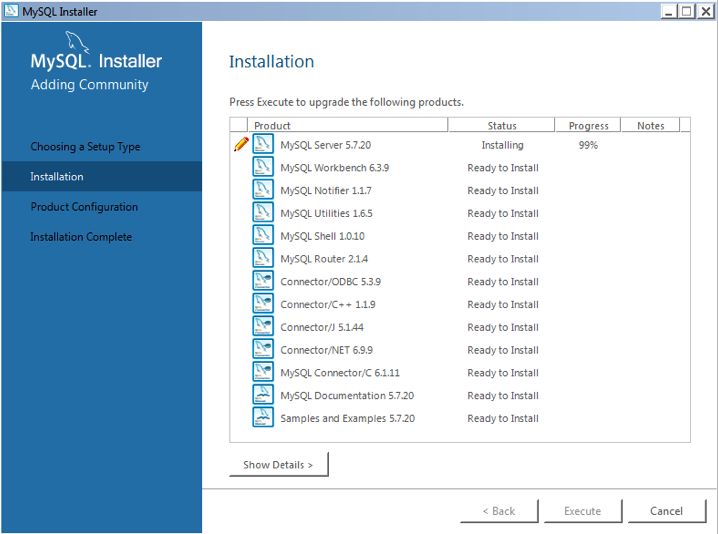


图 11

5、产品配置

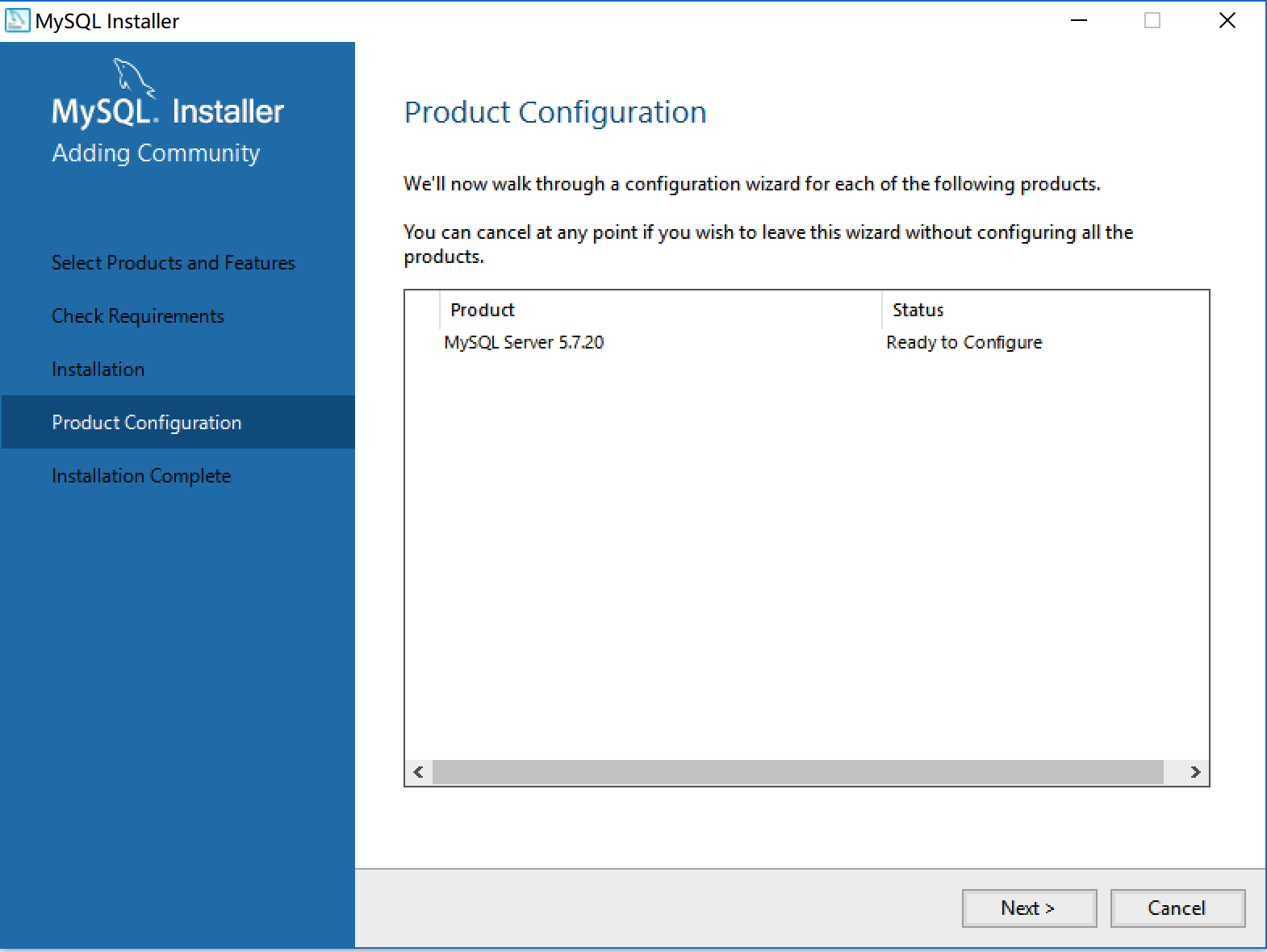


图 12

6、类型及网络选择。选择默认项即可。

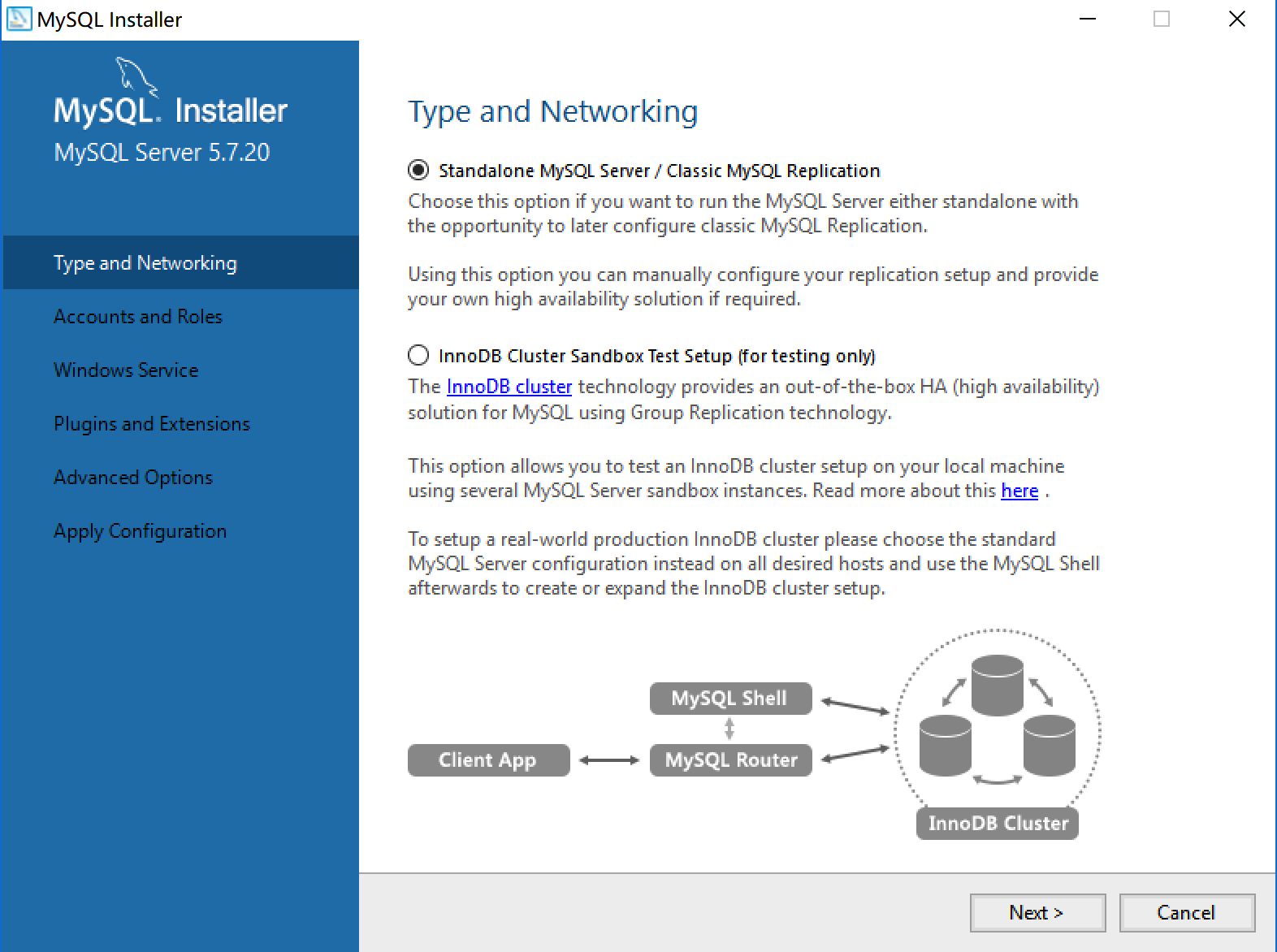


图 13

7、设置服务器配置类型。有三个使用类型，开发者机器、服务器机器、网络专用服务器。如果是个人，一般选择“开发者机器（Development Machine）”就可以了。设置端口号为3306。

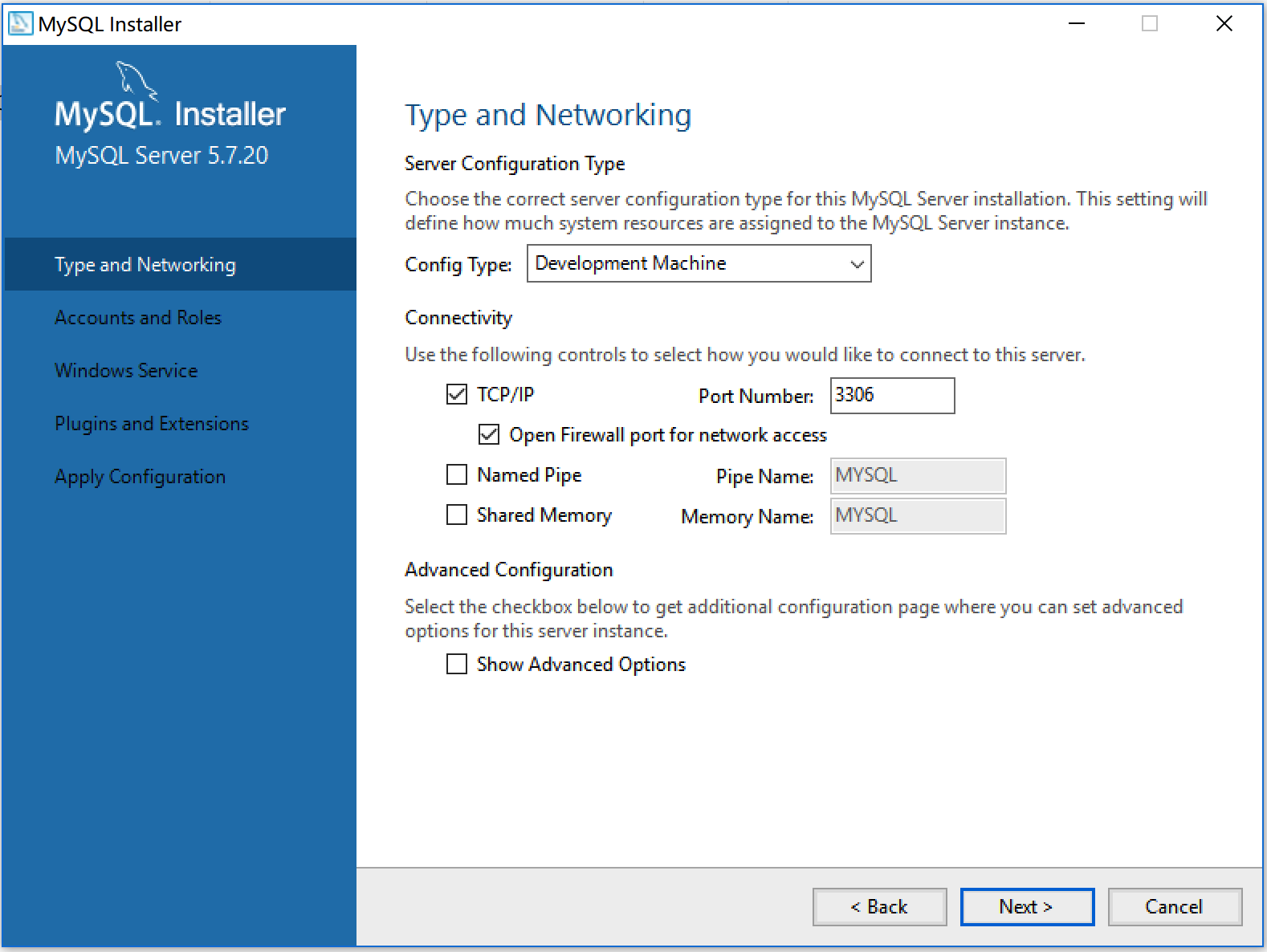


图 14

8、设置账号密码

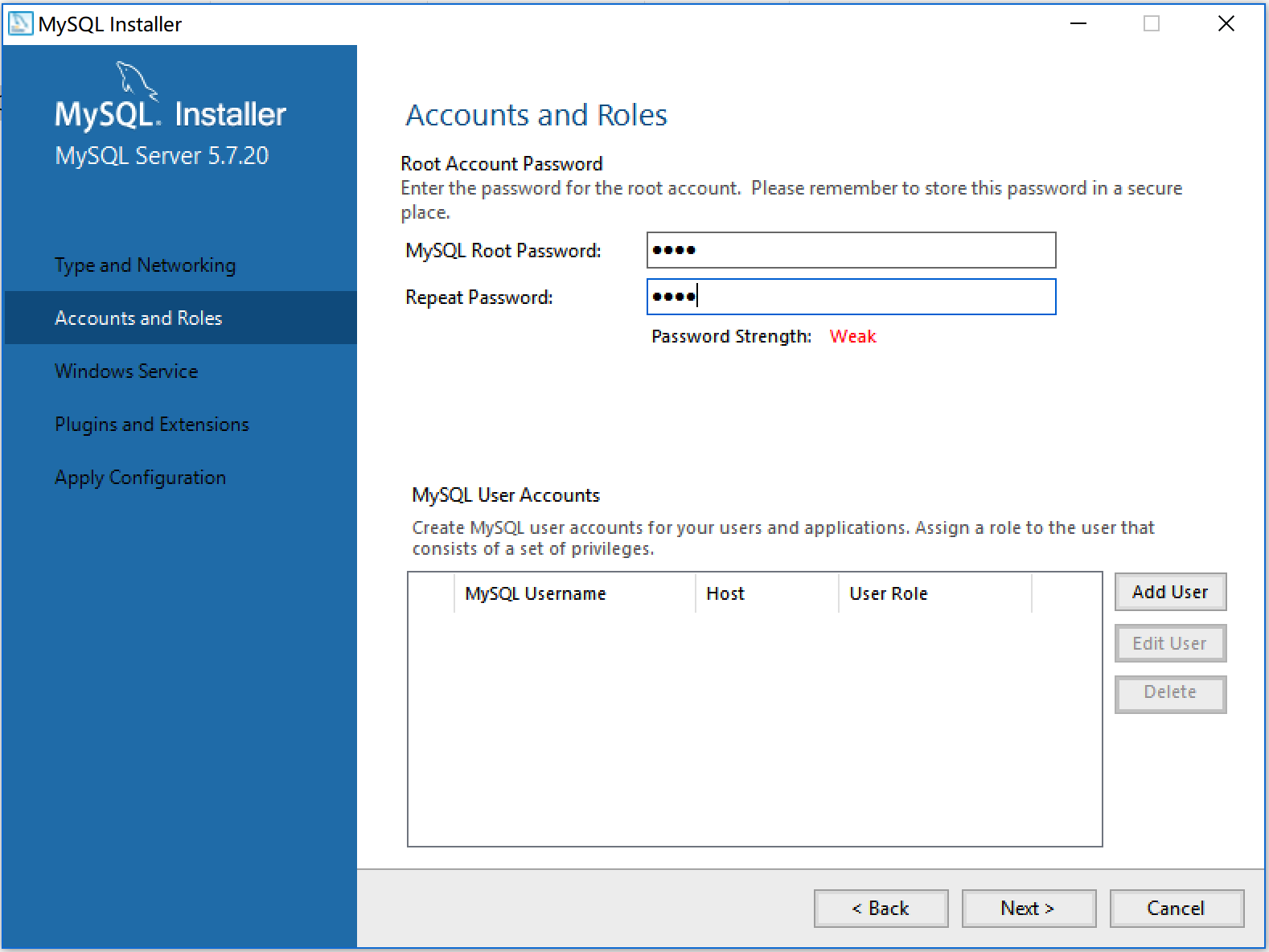


图 15

9、勾选“把MySQL设置成Windows服务”；

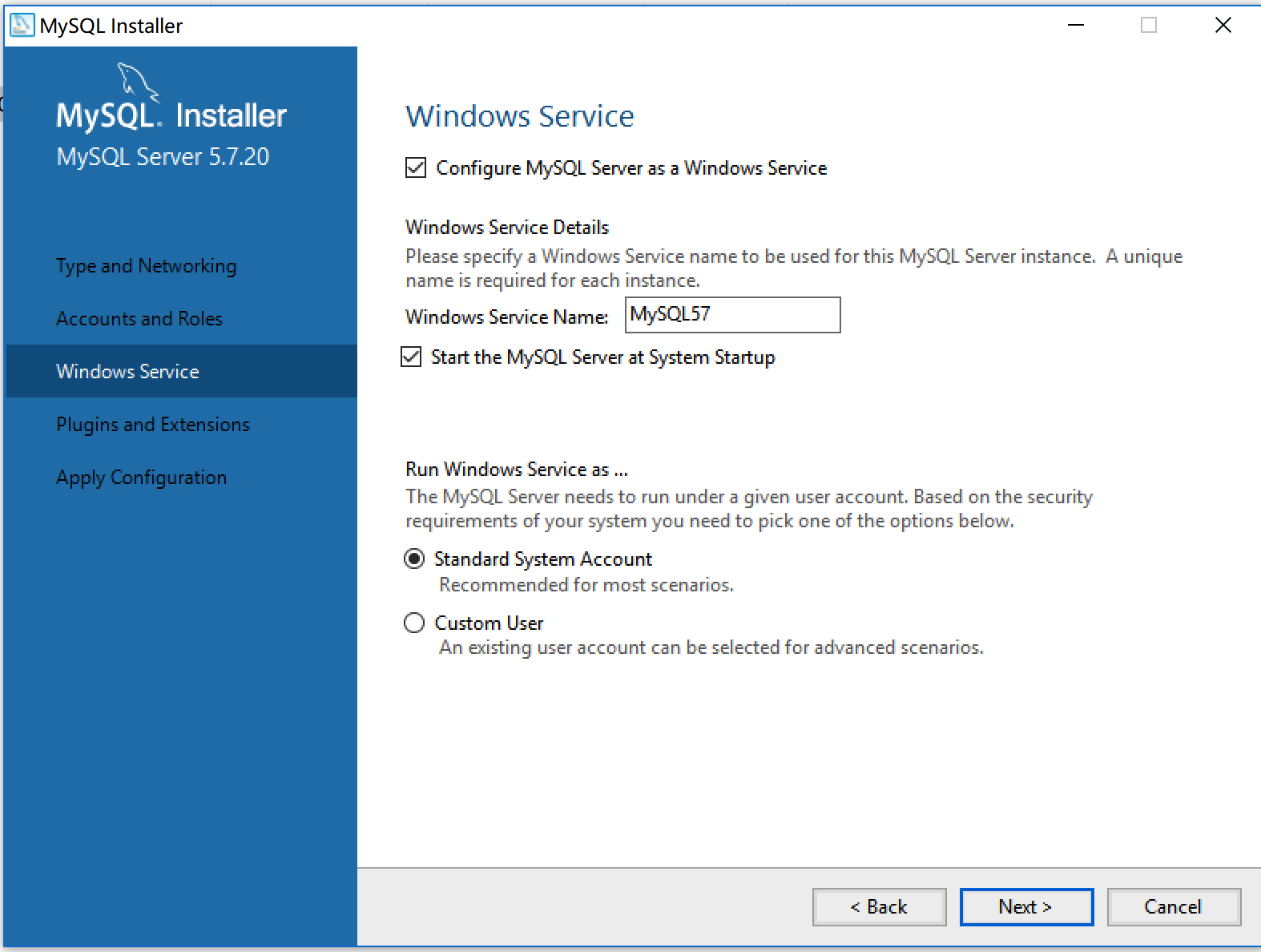


图 16

10、应用配置。配置完成会在前方显示绿色对勾。

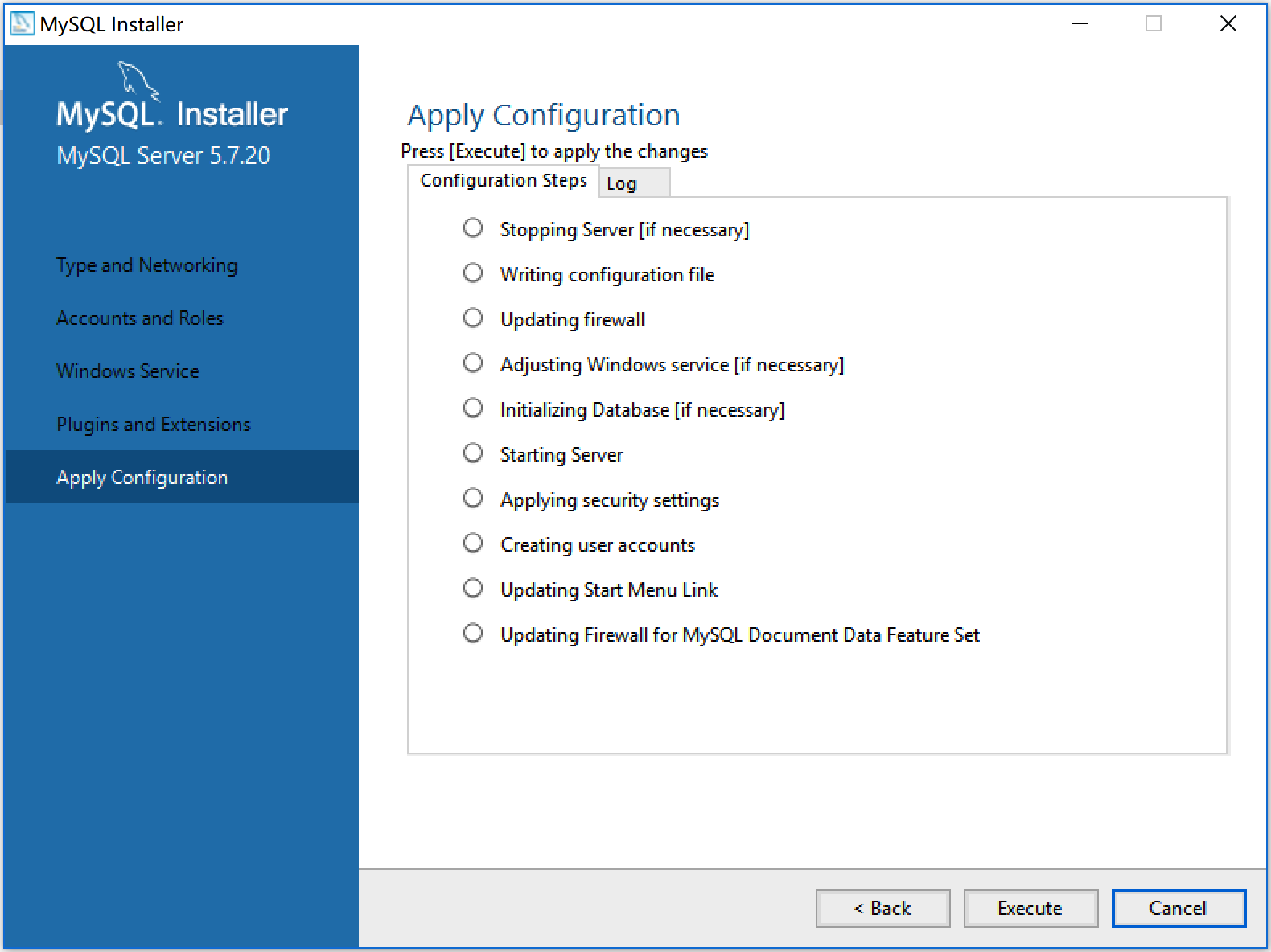


图 17

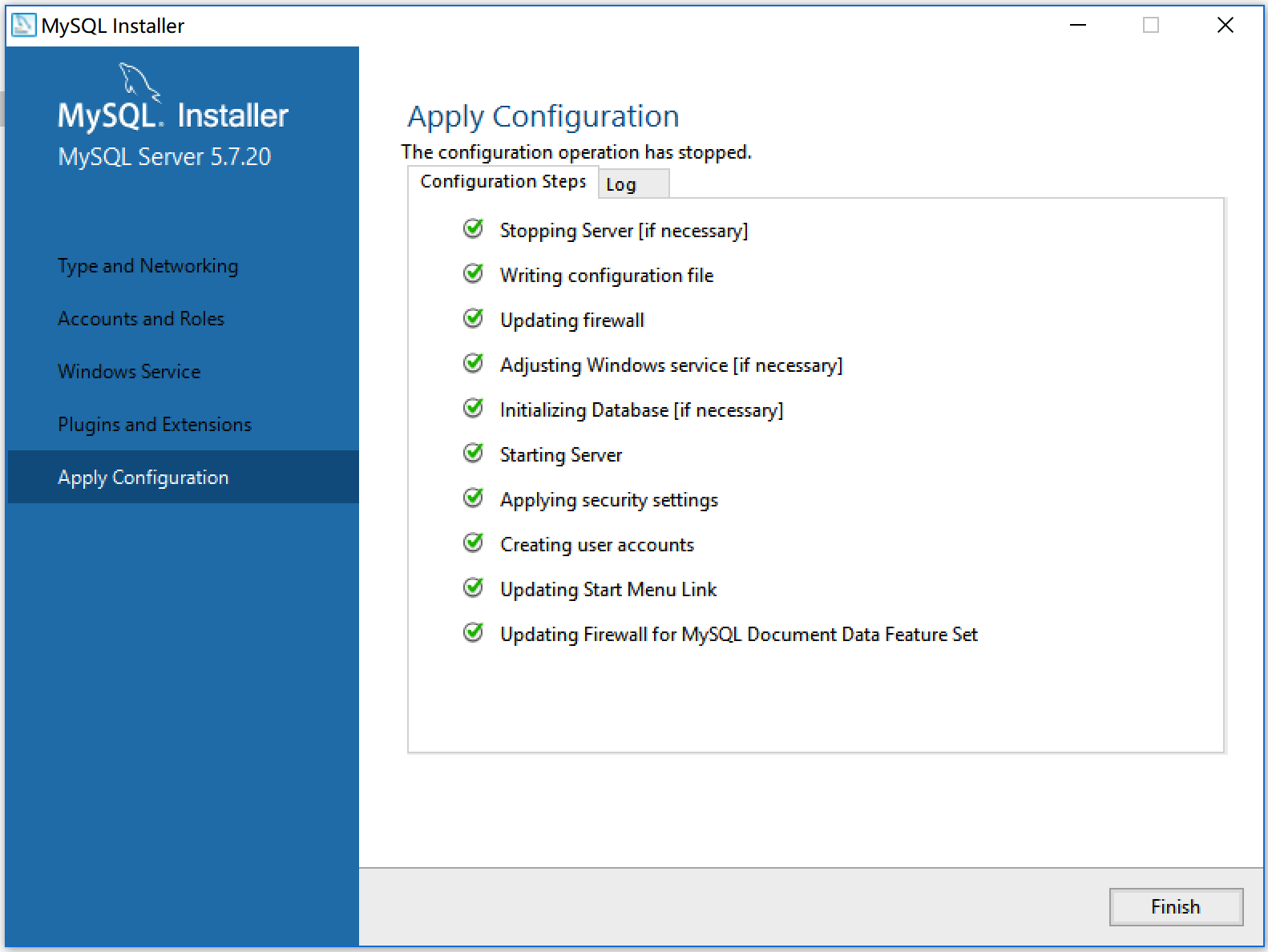


图 18

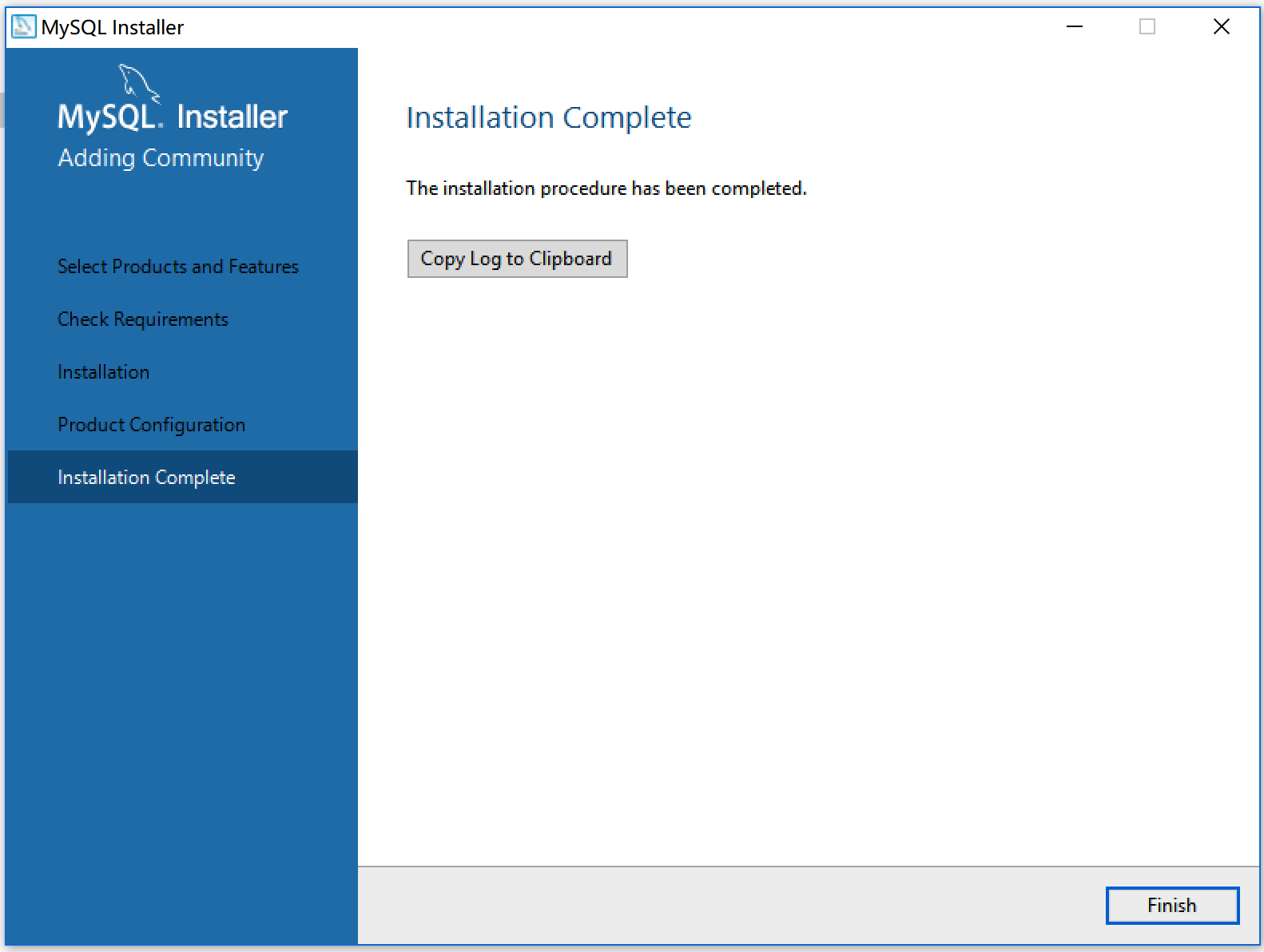


图 19

11、打开计算机“服务”，启动和停止MySQL服务。

### 安装后目录如下图所示

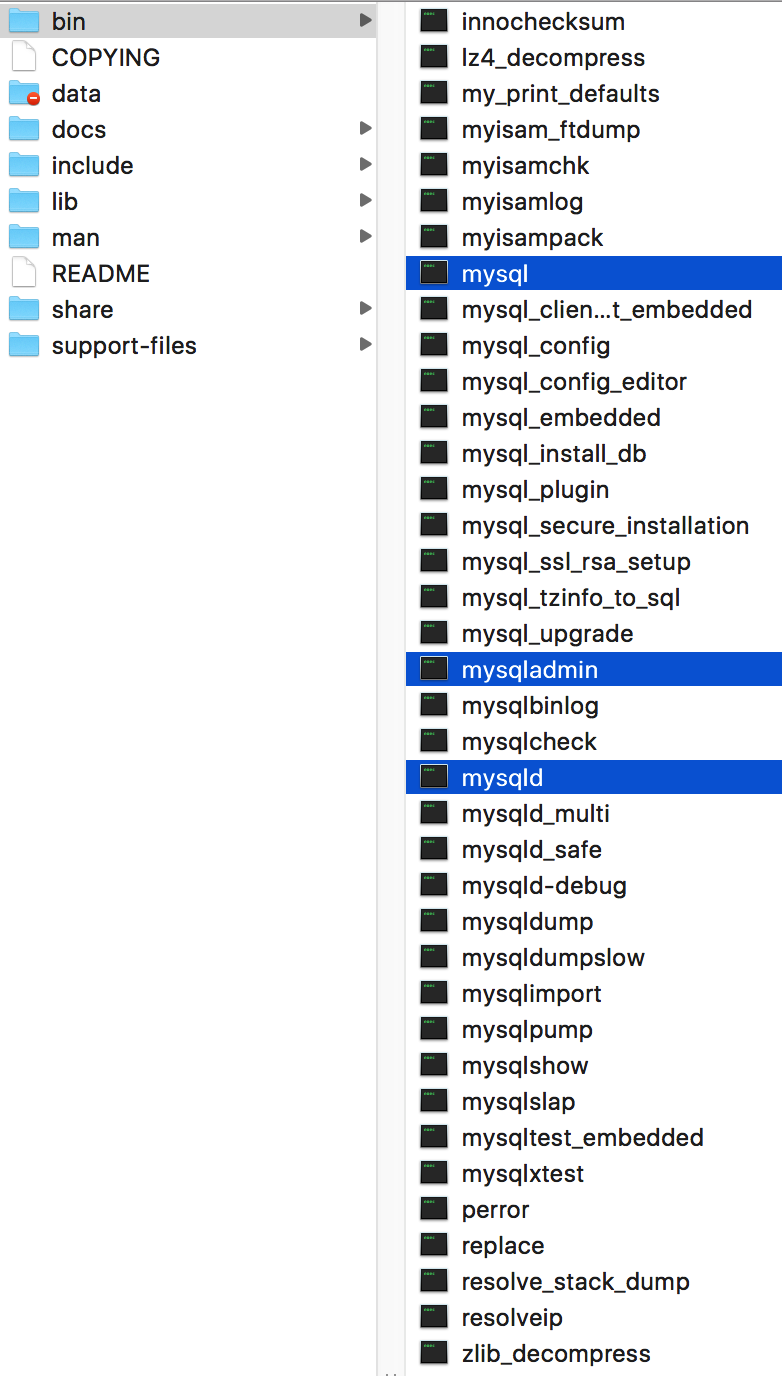


图 20

## 启动和停止MySQL服务

MySQL服务和MySQL数据库不同，MySQL服务是一系列的后台进程，而MySQL数据库是一系列的数据目录和数据文件； MySQL数据库必须在MySQL服务启动之后才可以进行访问。

### Windows系统下启动和停止MySQL服务

1、采用图形化方式启动和关闭MySQL

● 开始 → 控制面板 → 管理工具 → 服务

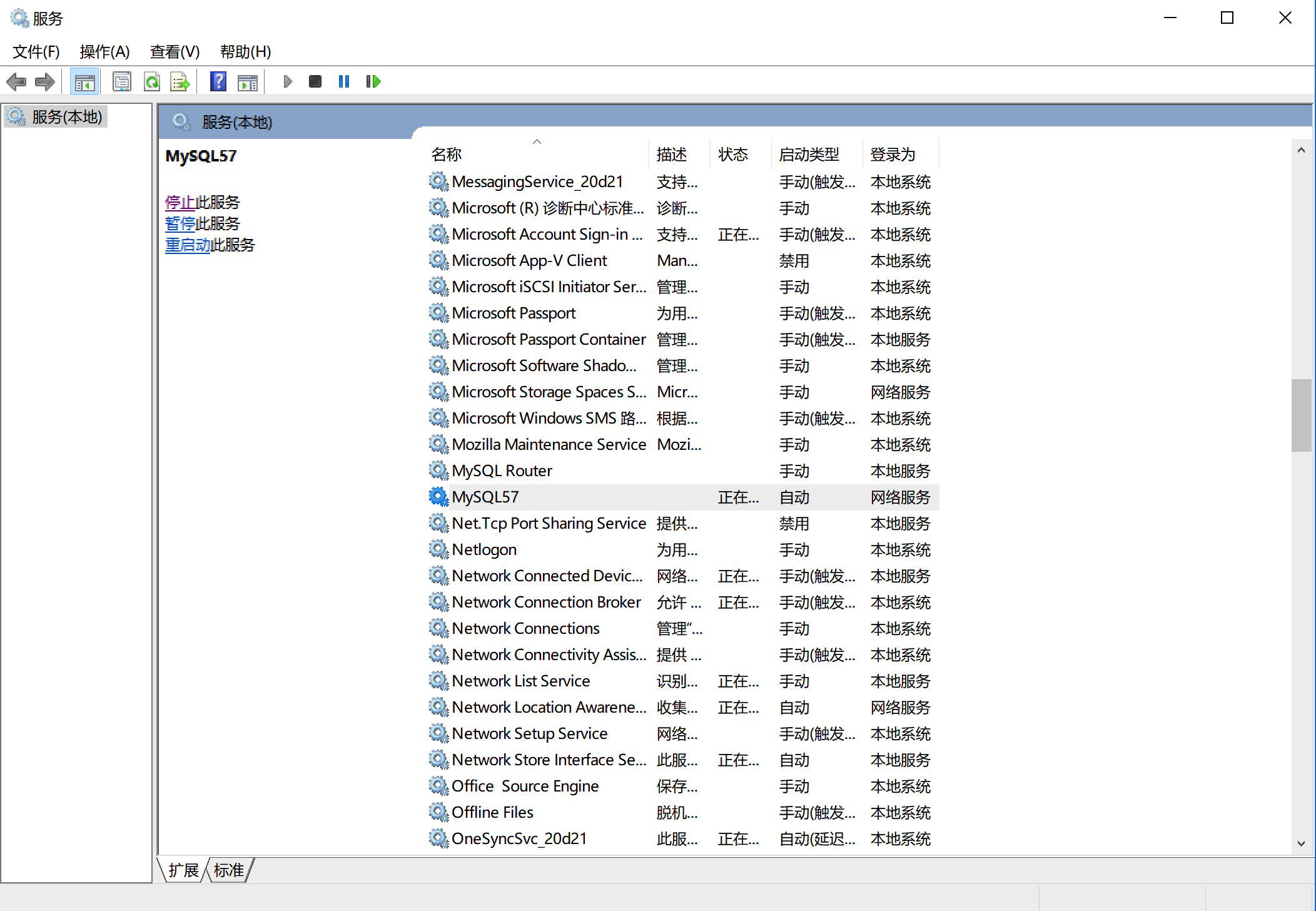


图 21

2、在DOS窗口下，通过命令行手工启动或关闭MySQL

● 需要先使用DOS命令进入mysql的安装目录下的bin目录中

○ cd C:\Program Files (x86)\MySQL\MySQL Server 5.7\bin

○ 如果配置环境变量，则无需进入到bin目录的步骤【环境变量配置见MySQL安装一节】

○ 设置环境变量的操作步骤

1）右键单击我的电脑->属性->高级系统设置(高级)->环境变量

点击系统变量下的新建按钮

输入变量名：MYSQL\_HOME

输入变量值：C:\Program Files (x86)\MySQL\MySQL Server 5.7

2）选择系统变量中的Path

点击编辑按钮

在变量值中添加变量值：%MYSQL\_HOME%\bin

注意是在原有变量值后面加上这个变量，用;隔开，不能删除原来的变量值

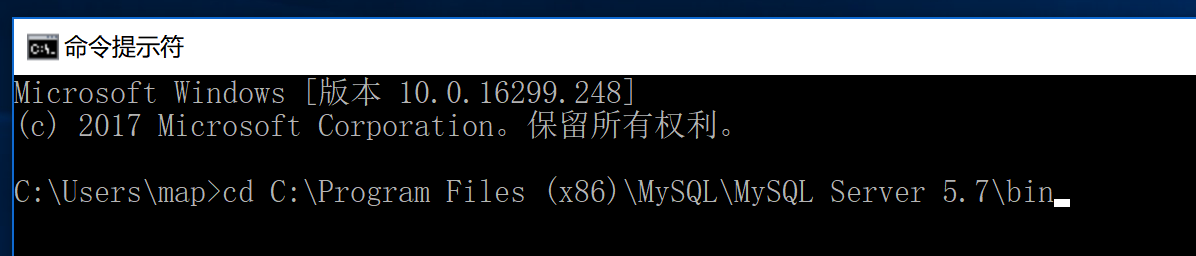


图 22

● 启动mysql数据库：在DOS命令窗口中输入 net start mysql

● 关闭mysql数据库：在DOS命令窗口中输入 net stop mysql

### 在终端连接数据库

● 在DOS命令窗口输入 mysql -hlocalhost -uroot -p回车，连接MySQL数据库

○ 其中-h表示服务器名，localhost表示本地，如果访问本地数据库，则该参数可以不写；

○ -u为数据库用户名，root是mysql默认用户名；

○ -p为密码，如果没有设置密码，该参数可以不写；

○ 如果设置了密码，可直接在-p后输入，比如：-p123456；

○ 如果设置了密码，但是用户没有写-p参数，则会提示Enter password，输入密码后回车即可。

● 成功连接到MySQL数据库后，DOS命令提示符为“mysql”;

○ mysql>

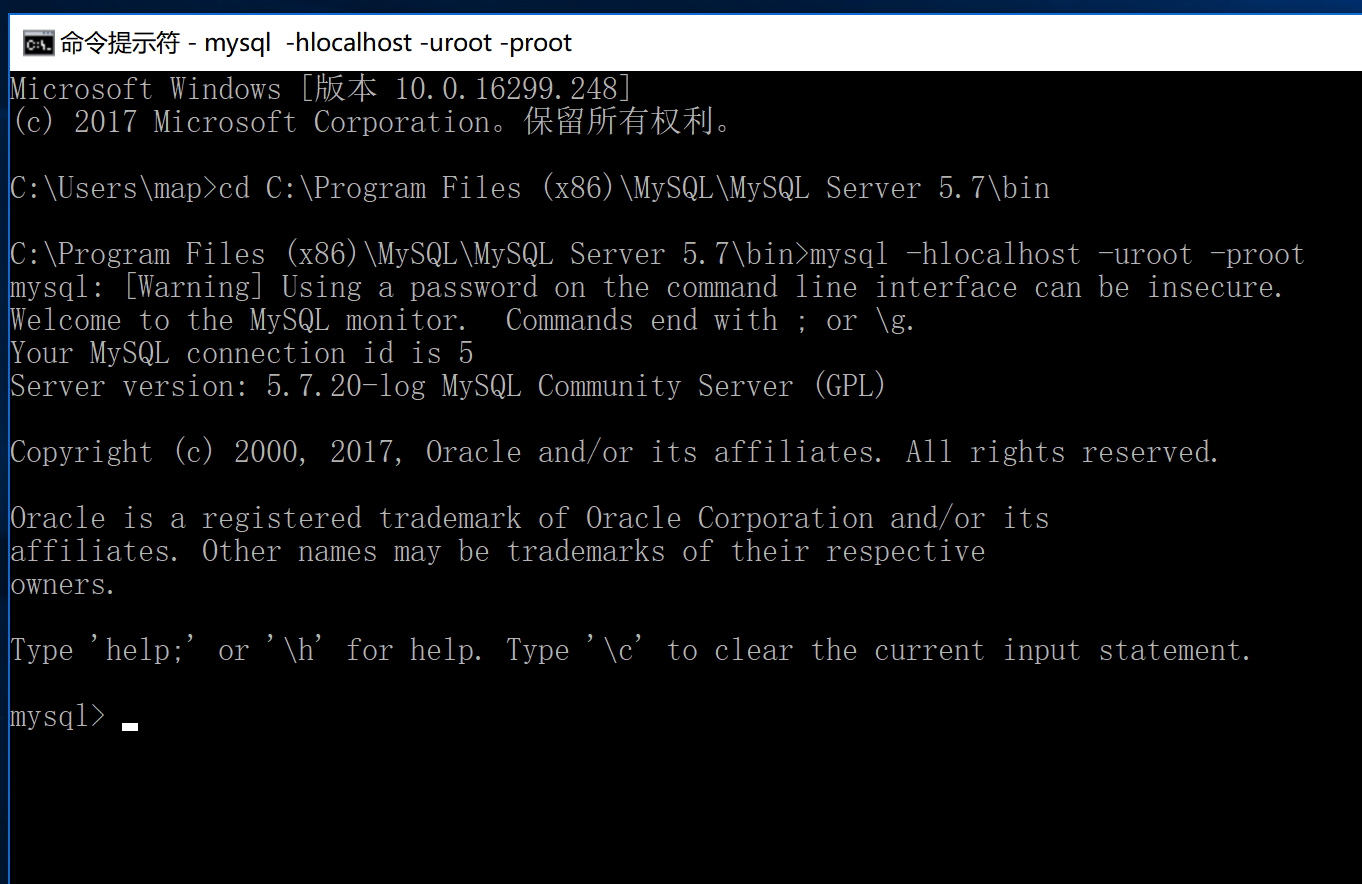


图 23

● 查看数据库版本。

○ mysql> select version();

● 查看当前时间。

○ mysql> select now();

● exit可以退出数据库。

○ mysql> exit;

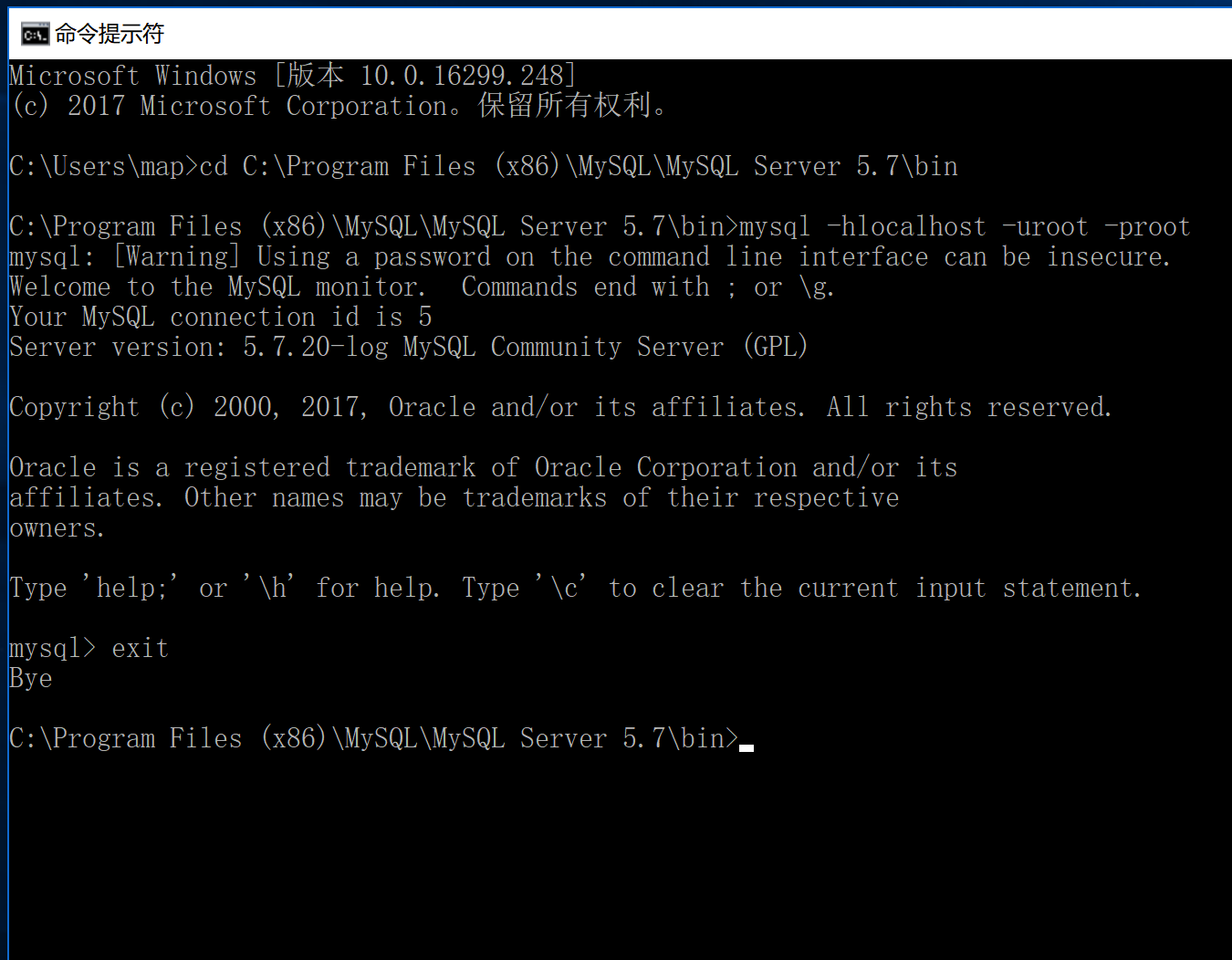


图 24

### MySQL数据库基本操作

1、SHOW DATABASES

○ 列出数据库管理系统中所有的数据库列表。

○ mysql> SHOW DATABASES;

○ 在安装MySQL时系统会自动创建几个数据库，一般都会有information\_schema、mysql数据库；

○ information\_schema：主要存储了系统中的一些数据库对象信息，比如用户表信息、列信息、权限信息、字符集信息、分区信息等；

○ mysql：存储了系统的用户权限信息；

mysql> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sys |

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

4 rows in set (0.00 sec)

2、USE 数据库名

○ 切换到要操作的数据库。使用该命令后所有SQL命令都只针对该数据库;

○ mysql> USE 数据库名;

mysql> use testdb;

Database changed

3、SHOW TABLES

○ 显示指定数据库的所有表，使用该命令前需要使用 use 命令来选择要操作的数据库;

○ mysql> SHOW TABLES;

mysql> show tables;

Empty set (0.00 sec)

4、DESC 表名称

○ 显示表定义时的结构。包括该表中每列（字段）的名称、该列的数据类型、该列的约束条件等。

○ mysql> DESC 表名称;

mysql> use mysql

mysql> desc servers;

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

| Field | Type | Null | Key | Default | Extra |

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

| Server\_name | char(64) | NO | PRI | | |

| Host | char(64) | NO | | | |

| Db | char(64) | NO | | | |

| Username | char(64) | NO | | | |

| Password | char(64) | NO | | | |

| Port | int(4) | NO | | 0 | |

| Socket | char(64) | NO | | | |

| Wrapper | char(64) | NO | | | |

| Owner | char(64) | NO | | | |

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

9 rows in set (0.00 sec)

### Mac系统下启动和停止MySQL服务

1、进入系统偏好设置



图 25

2、启动MySQL设置，弹出MySQL对话框

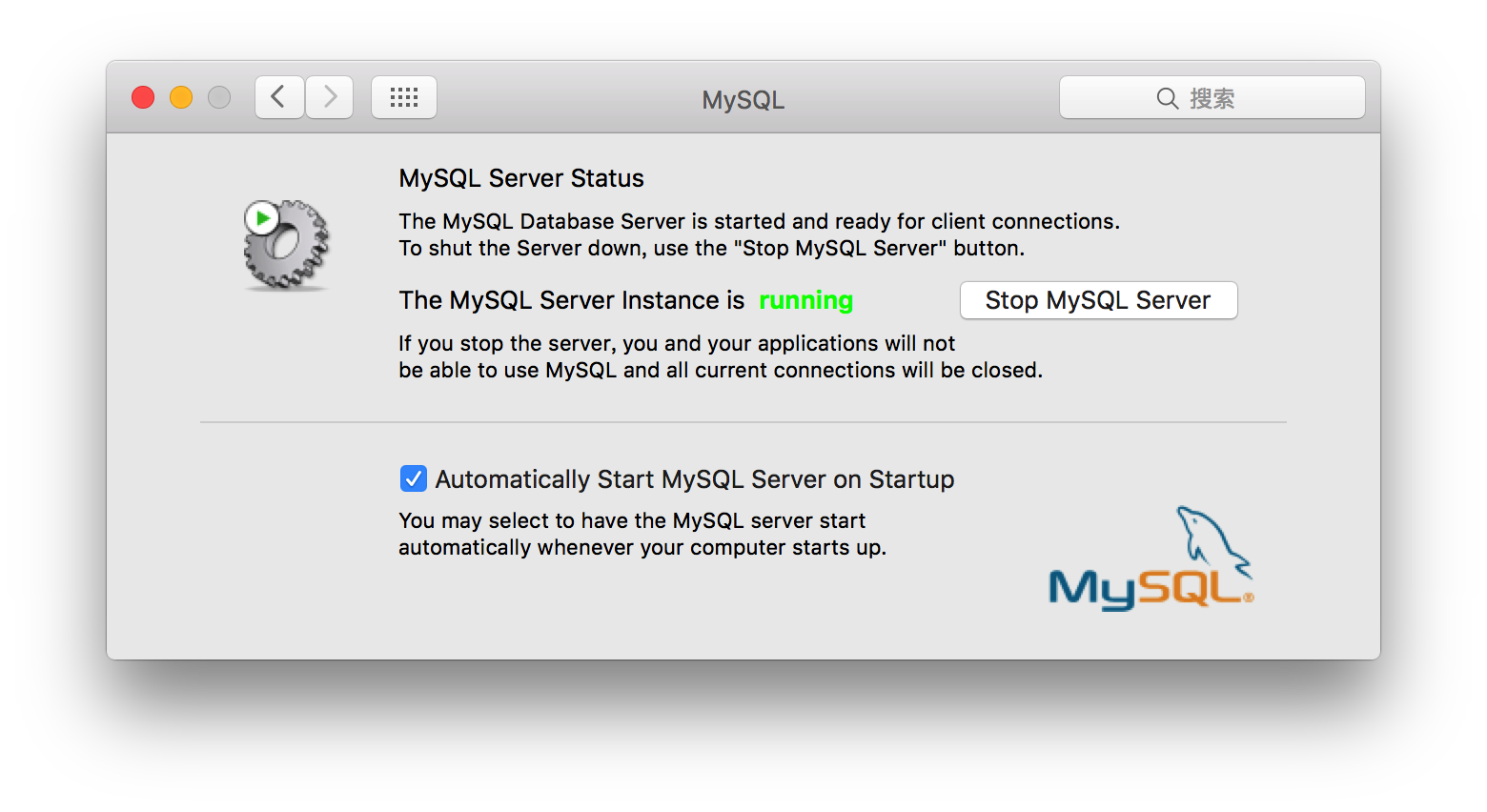


图 26

## 客户端管理工具Navicat

### Navicat数据库管理工具简介

1、Navicat premium (导航猫数据库管理软件 - 高配版) 是一款数据库管理工具，是一个可多重连线资料库的管理工具，它可以让你以单一程式同时连线到 MySQL、SQLite、Oracle 及 PostgreSQL 资料库，让管理不同类型的资料库更加的方便。

2、Navicat for MySQL是一套管理和开发MySQL的理想解决方案。这个功能齐备的前端软件为数据库管理、开发和维护提供了直观而强大的图形界面，给MySQL新手以及专业人士提供了一组全面的工具。

### Navicat for MySQL 软件下载及安装



图 27



图 28

### Navicat软件界面介绍

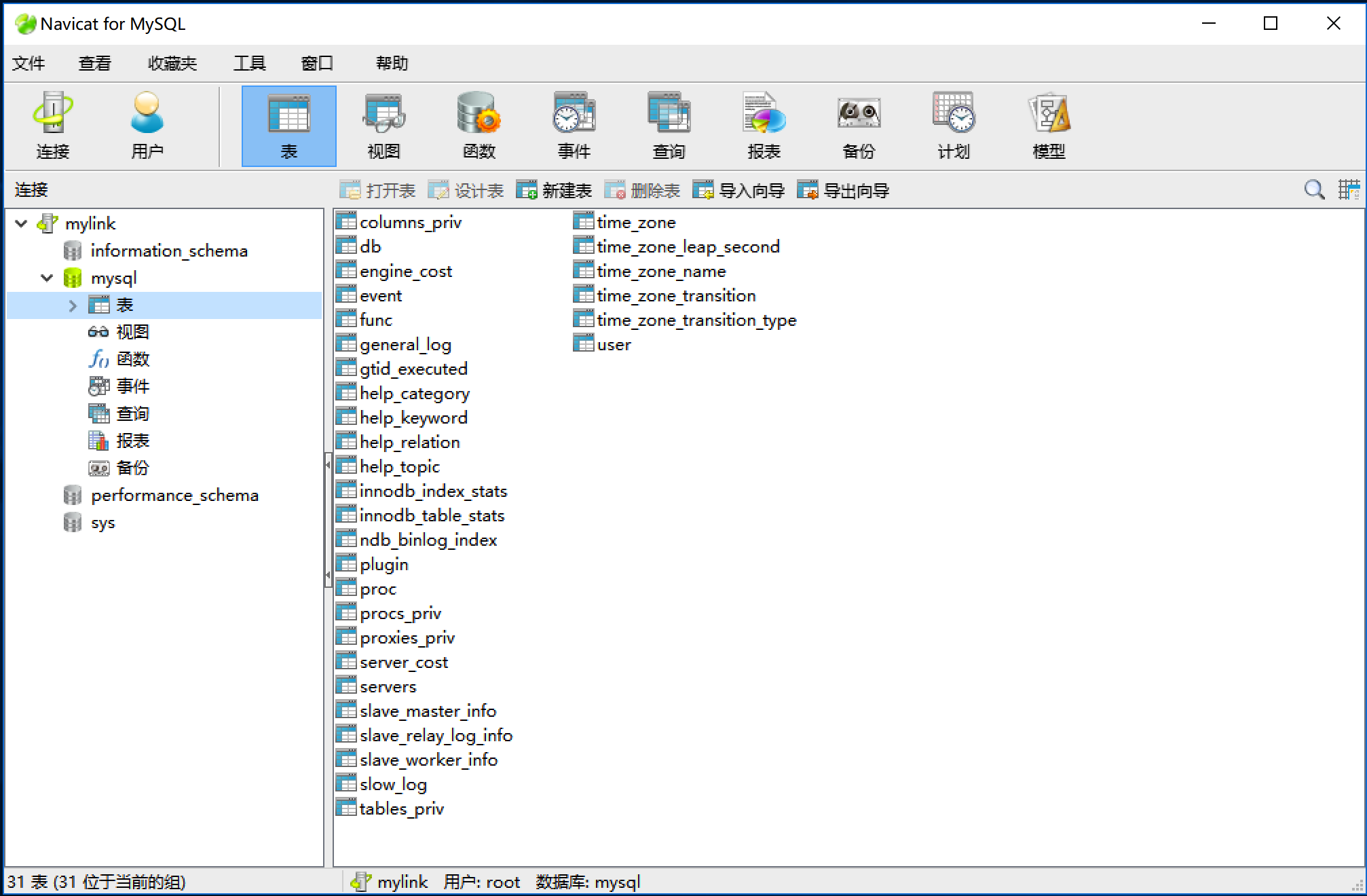


图 29

## MySQL数据类型

### MySQL支持的数据类型

MySQL提供了多种数据类型，主要包括数值型、字符串类型、日期和时间类型。

1、数值类型

● 整数类型：tinyint、smallint、mediumint、int（integer）、bigint

● 浮点数类型：float、double 、dec（decimal）

● 位类型：bit

● 最常用的是: int(整数)、double 、dec（decimal）

备注：int是integer的同义词，dec是decimal的同义词。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 大小 | 范围（有符号） | 范围（无符号） | 用途 |
| TINYINT | 1 字节 | (-128，127) | (0，255) 2的8次方 | 微小整数值 |
| SMALLINT | 2 字节 | (-32 768，32 767) | (0，65 535) 2的16次方 | 小整数值 |
| MEDIUMINT | 3 字节 | (-8 388 608，8 388 607) | (0，16 777 215) 2的24次方 | 中整数值 |
| INT或INTEGER | 4 字节 | (-2 147 483 648，2 147 483 647) | (0，4 294 967 295) 2的32次方 | 大整数值 |
| BIGINT | 8 字节 | (-9 233 372 036 854 775 808，9 223 372 036 854 775 807) | (0，18 446 744 073 709 551 615) | 极大整数值 |
| FLOAT | 4 字节 | (-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38) | 0，(1.175 494 351 E-38，3.402 823 466 E+38) | 单精度  浮点数值 |
| DOUBLE | 8 字节 | (-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 双精度  浮点数值 |
| DECIMAL | 对DECIMAL(M,D) ，如果M>D，为M+2否则为D+2 | 依赖于M和D的值 | 依赖于M和D的值 | 小数值 |

2、日期时间类型

● 表示时间值的日期和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。

● 最常用: datatime类型.

● 每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 大小  (字节) | 范围 | 格式 | 用途 |
| DATE | 3 | 1000-01-01/9999-12-31 | YYYY-MM-DD | 日期值 |
| TIME | 3 | '-838:59:59'/'838:59:59' | HH:MM:SS | 时间值或持续时间 |
| YEAR | 1 | 1901/2155 | YYYY | 年份值 |
| DATETIME | 8 | 1000-01-01 00:00:00/9999-12-31 23:59:59 | YYYY-MM-DD HH:MM:SS | 混合日期和时间值 |
| TIMESTAMP | 4 | 1970-01-01 00:00:00/2038  结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07 | YYYYMMDD HHMMSS | 混合日期和时间值，时间戳 |

3、字符串类型

● 字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。

● 最常用的: char、varchar和text类型.

● CHAR和VARCHAR类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

● BINARY和VARBINARY类类似于CHAR和VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

● BLOB是一个二进制大对象，可以容纳可变数量的数据。有4种BLOB类型：TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB。它们只是可容纳值的最大长度不同。

● 有4种TEXT类型：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT。这些对应4种BLOB类型，有相同的最大长度和存储需求。

|  |  |  |
| --- | --- | --- |
| 类型 | 大小 | 用途 |
| CHAR | 0-255字节 | 定长字符串 |
| VARCHAR | 0-65535 字节 | 变长字符串 |
| TINYBLOB | 0-255字节 | 不超过 255 个字符的二进制字符串 |
| TINYTEXT | 0-255字节 | 短文本字符串 |
| BLOB | 0-65 535字节 | 二进制形式的长文本数据 |
| TEXT | 0-65 535字节 | 长文本数据 |
| MEDIUMBLOB | 0-16 777 215字节 | 二进制形式的中等长度文本数据 |
| MEDIUMTEXT | 0-16 777 215字节 | 中等长度文本数据 |
| LONGBLOB | 0-4 294 967 295字节 | 二进制形式的极大文本数据 |
| LONGTEXT | 0-4 294 967 295字节 | 极大文本数据 |

4、总结常用的类型：

● int：整型

● double：浮点型，例如double(5,2)表示最多5位，其中必须有2位小数，即最大值：999.99

● decimal：浮点型，不会出现精度缺失问题，比如金钱。

● char：固定长度字符串类型：最大长度：char(255)

● varchar：可变长度字符串类型：最大长度：varchar(65535)

● text(clob)：字符串类型，存储超大文本。

● blob：字节类型，最大4G

● date：日期类型，格式为：yyyy-MM-dd

● time：时间类型：格式为：hh:mm:ss

● timestamp：时间戳

● datatime：日期+时间，格式为：yyyy-MM-dd hh:mm:ss

## SQL基础入门

### SQL语言简介

1、概念

● 结构化查询语言(Structured Query Language)简称SQL(发音：/ˈes kjuː ˈel/ "S-Q-L")，是一种特殊目的的编程语言；

● SQL是最重要的关系型数据库操作语言，用于存取数据以及查询、更新和管理关系数据库系统；

● SQL的影响力已经超出数据库领域，得到其他领域的重视和采用，如人工智能领域的数据检索，第四代软件开发工具中嵌入SQL的语言等。

2、发展历史

● 在20世纪70年代初，IBM公司在研制关系型数据库管理系统System R时，研制出一套规范语言-SEQUEL （发音[ˈsi:kwəl]）（Structured English Query Language），并在1976年11月公布新版本，叫SEQUEL/2。1980年改名为SQL。

● 1979年ORACLE公司首先应用SQL，IBM公司在DB2和SQL/DS数据库系统中也实现了SQL。

● 1986年10月，美国国家标准局ANSI采用SQL作为关系数据库管理系统的标准语言。接着，国际标准化组织（ISO）颁布了SQL正式国际标准。

● 1989年4月，ISO提出了具有完整性特征的SQL89标准，1992年11月又公布了SQL92标准，在此标准中，把数据库分为三个级别：基本集、标准集和完全集。

● 目前主要的关系数据库管理系统支持SQL89标准。

● 不过各种通行的数据库系统在其实践过程中都对SQL规范作了某些编改和扩充。所以，不同数据库系统之间的SQL略有不同，不能完全相互通用。

### SQL分类

完整的结构化查询语言包含6个部分，或者说SQL语句有以下六个类别。

1、数据查询语言（DQL:Data Query Language）

● 也称为“数据检索语句”，用以从表中获得数据；

● 查询语句的关键字是：SELECT。select语句是所有SQL中最重要的语句；

● DQL其它常用的保留字有：WHERE、ORDER BY、GROUP BY和HAVING。这些DQL保留字常与其他类型的SQL语句一起使用。

2、数据操作语言（DML：Data Manipulate Language）

● 关键字是：INSERT、UPDATE和DELETE。它们分别用于添加，修改和删除表中的行。

3、数据定义语言（DDL：Data Define Language）

● 关键字包括：CREATE、DROP、ALTER，可以用于创建数据库（CREATE DATABASE） 、删除数据库（DROP DATABASE）、创建表（CREAT TABLE）、删除表 （ DROP TABLE）以及修改表结构（ALTER TABLE）等。

4、事务处理语言（TPL：Transaction Process Language）

● 事务主要用于同时执行多条SQL语句的情况，事务能确保多条执行的SQL语句能同时执行成功，一旦有其中任何一条执行失败，则同时都不执行，从而保证数据一致性。当批量执行数据操作时，使用事务可以提高执行效率。TPL语句包括开始事务（BEGIN TRANSACTION）、执行事务（COMMIT）和事务回滚（ROLLBACK）。多条被执行的SQL语句包含在开始事务和执行事务语句之间，当出现异常就执行事务回滚，从而保证所有的SQL语句都不执行。

5、数据控制语言（DCL：Data Control Language）

● 该语句通过GRANT或REVOKE获得许可，确定单个用户和用户组对数据库对象的访问。DBA(Database Administrator，数据库管理员)会更多使用DCL语句。

6、指针控制语言（CCL：：Cursor Control Language）

● 该类型语句，比如：DECLARE CURSOR，FETCH INTO和UPDATE WHERE CURRENT用于对一个或多个表单独行的操作。

【备注：】

● 学习SQL的主要重心应该放在DQL和DML语句上，尤其是DQL是重点及难点；

● CRUD：是指在做计算处理时的增加(Create)、查询(Retrieve)（重新得到数据）、更新(Update)和删除(Delete)几个单词的首字母简写。主要被用在描述数据库的基本操作。其实就是DQL + DML。

### DDL语句

包括：创建数据库、删除数据库、创建表、删除表、修改表。

1、创建数据库

● 语法结构：CREATE DATABASE 数据库名;

例如：

mysql> create database testdb;

Query OK, 1 row affected (0.00 sec)

Query OK表示执行成功。

如果创建已经存在的数据库，则会提示：

ERROR 1007 (HY000): Can't create database 'testdb'; database exists

表示创建数据库失败，因为该库已经存在。

说明:

● 数据库名不要使用中文

● 由于数据库中将来会存储一些非ascii字符, 所以务必指定字符编码, 一般都是指定utf-8编码

● CHARSET 选择 utf8 COLLATION 选择 utf8\_general\_ci

● mysql中字符集是utf8，不是utf-8。

2、删除数据库

● 语法结构：DROP DATABASE 数据库名;

mysql> drop database testdb;

Query OK, 0 rows affected (0.00 sec)

备注：数据库删除后，该库中所有的表数据都会全部删除，所以删除前一定要仔细检查并做好备份工作。

3、创建表

● 语法结构：

CREATE TABLE 表名 (列名称1 数据类型1 列约束条件1 , 列名称2 数据类型2 列约束条件2 , 列名称3 数据类型3 列约束条件3 ...)

列名称、数据类型是必选项，而约束条件是可选项。

● 选择数据库引擎。有两种选择：InnoDB或MyISAM。如果要支持事务，需要选择InnoDB，MyISAM不支持事务；如果对该表的绝大多数操作都只是查询数据，而很少增、删、改，可以使用MyISAM，如果读写比较频繁，那么就得使用InnoDB数据库引擎。

例如：创建一个“员工表”，名字为emp，其中包括“姓名ename” ， “雇佣日期hiredate”，“薪水salary” ， “部门编号depno”。

//创建数据库testdb

mysql> create database testdb;

Query OK, 1 row affected (0.01 sec)

//切换到数据库testdb

mysql> use testdb;

Database changed

//在数据库testdb中创建表emp

mysql> CREATE TABLE emp(ename varchar(20) , hiredate date , salary decimal(10,2) , deptno tinyint(2)) ;

Query OK, 0 rows affected (0.02 sec)

//展示testdb中所有的表

mysql> show tables;

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

| Tables\_in\_testdb |

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

| emp |

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

1 row in set (0.00 sec)

//展示emp表的结构

mysql> desc emp;

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

| Field | Type | Null | Key | Default | Extra |

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

| ename | varchar(10) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

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

4 rows in set (0.01 sec)

4、删除表

● 语法结构：DROP TABLE 表名;

mysql> drop table emp2;

Query OK, 0 rows affected (0.01 sec)

5、修改表

(1)、更改表名

● 语法结构：ALTER TABLE 原表名 RENAME 新表名;

mysql> alter table emp0 rename emp1;

Query OK, 0 rows affected (0.01 sec)

(2）、增加表字段（表中增加新列）

● 语法结构：ALTER TABLE 表名 ADD [COLUMN] 列名 新列的数据类型;

● 关键字 “COLUMN” 为可选项，可以省略不写。

mysql> ALTER TABLE emp1 ADD COLUMN age tinyint(3);

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

(3）、删除表字段

● 语法结构：ALTER TABLE 表名 DROP [COLUMN] 列名;

● 关键字 “COLUMN” 为可选项，可以省略不写。

mysql> ALTER TABLE emp1 ADD tel varchar(15);

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE emp1 DROP tel ;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

(4）、修改表字段类型

● 语法结构：ALTER TABLE 表名 MODIFY [COLUMN] 列名 列数据类型;

● 关键字 “COLUMN” 为可选项，可以省略不写。

//展示修改前表的结构类型

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(10) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

| age | int(3) | YES | | NULL | |

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

6 rows in set (0.00 sec)

//修改ename字段的类型

mysql> ALTER TABLE emp1 MODIFY ename varchar(20);

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

//展示修改后表的结构类型

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(20) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

| age | int(3) | YES | | NULL | |

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

6 rows in set (0.00 sec)

(5）、修改表字段名称

● 语法结构：ALTER TABLE 表名 CHANGE [COLUMN] 原列名 新列明 新列的数据类型;

● 关键字 “COLUMN” 为可选项，可以省略不写。

//修改前表结构

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(20) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

| age | int(3) | YES | | NULL | |

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

6 rows in set (0.00 sec)

//修改表的字段名称和类型

mysql> ALTER TABLE emp1 CHANGE age age1 int(4) ;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

//展示修改后表的新字段名称和类型

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(20) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

| age1 | int(4) | YES | | NULL | |

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

6 rows in set (0.00 sec)

(6）、修改表字段排列顺序

● 在ADD 、 MODIFY 、 CHANGE 三个关键字的用法中，都有一个可选项first|after 列名称；

● 这个可选项用来修改字段在表中的位置，默认ADD新增的字段在表的最后位置，而CHANGE和MODIFY则不改变字段的位置。

● 借助ADD 、 MODIFY 、 CHANGE 三个关键字可以修改表字段的排列顺序。

//新增字段tel，在ename字段的后面

mysql> ALTER TABLE emp1 ADD tel varchar(15) AFTER ename;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

//展示修改后表结构

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(20) | YES | | NULL | |

| tel | varchar(15) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

| age1 | int(4) | YES | | NULL | |

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

7 rows in set (0.00 sec)

//修改age1字段为age，将该字段放到ename后面

mysql> ALTER TABLE emp1 CHANGE age1 age int(3) AFTER ename;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

//展示修改后表结构

mysql> desc emp1;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| ename | varchar(20) | YES | | NULL | |

| age | int(3) | YES | | NULL | |

| tel | varchar(15) | YES | | NULL | |

| hiredate | date | YES | | NULL | |

| salary | decimal(10,2) | YES | | NULL | |

| deptno | int(2) | YES | | NULL | |

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

7 rows in set (0.00 sec)

### DML语句基本用法

1、插入数据

● 插入新数据：INSERT

● 语法：INSERT INTO 表名(字段1 , 字段2 ... ) VALUES(值1 , 值2 ...);

● 例如：

○ INSERT INTO emp(ename , age , hiredate , salary , deptno) VALUES('Steven', 30, '2015-12-15' , 40000 , 1);

2、更新记录

● 更新数据：UPDATE

● 语法：UPDATE 表名 SET 字段1 = 值1 , 字段2=值2 ... WHERE 条件;

● 例如：

○ UPDATE emp SET ename='StevenWang' , age=31 WHERE ename='Steven';

3、删除记录

● 删除数据：DELETE

● 语法：DELETE FROM 表名 WHERE 条件

● 例如：

○ DELETE FROM emp WHERE ename='StevenWang';

### DQL语句基本用法

1. 查询所有记录

● 查询数据：SELECT

● 语法：SELECT \* FROM 表名

● 例如：

○ SELECT \* FROM emp;

○ SELECT ename , age , salary FROM emp;

2.按照一定的条件查找

● 查询数据：SELECT

● 语法：SELECT \* FROM 表名 WHERE 条件

● 例如：

○ SELECT \* FROM emp WHERE salary>30000;

3、SQL高级查询

● sql运算符

● sql函数的用法

● 不重复记录查询（distinct）

● 排序和限制查询（order by 、 limit）

● 分组聚合查询（group by 、 count() sum() avg() max() min()）

● 多表连接查询(inner join left join right join )

● 子查询(in not in exists any all)

● 记录联合查询(union , union all)

● 给表和字段起别名的用法

## MySQL中运算符

mysql支持多种类型运算符。主要有：算术运算符、比较运算符、逻辑运算符、位运算符四种。

### 算术运算符

算术运算符包括：+ 、 - 、 \* 、 / 或者div 、% 或者mod。作用分别是加法、减法、乘法、除法、取余数。例如：

|  |
| --- |
| mysql> select 0.1+0.2, 0.1-0.2 , 0.1\*0.2, 0.1/0.2,1%2;  +---------+---------+---------+---------+-----+  | 0.1+0.2 | 0.1-0.2 | 0.1\*0.2 | 0.1/0.2 | 1%2 |  +---------+---------+---------+---------+-----+  | 0.3 | -0.1 | 0.02 | 0.50000 | 1 |  +---------+---------+---------+---------+-----+  1 row in set (0.00 sec) |

在八维电子商城案例中，以上算术运算符曾用在商品表中。当售出一件商品后，商品总数减去一个，而已经售出数量增加一个。算术运算符多用在update及insert语句中，而很少用在select或delete语句。示例SQL语句如下：

|  |
| --- |
| mysql> update tb\_goods set goods\_count=goods\_count-1 , sold\_count=sold\_count+1 where id=1;  1 row in set (0.00 sec) |

### 比较运算符

比较运算符中有以下六类：

* + 比较数值大小的：= 、<>或!= 、< 、 > 、 <= 、 > 、>= ；
  + 指定范围的：between and 、 not between and；
  + 指定集合的：in 、 not in ；
  + 是否为空值的：is null 、 is not null ；
  + 通配符匹配字符的：like 、not like
  + 正则表达式匹配字符的：regexp 或 rlike。

1. null不能用于“=”比较，也不能用于“<>”比较。如果想查找某个字段为null或不为null的记录，不可以写 字段=null 或者 字段<>null。只能使用IS NULL 或者 IS NOT NULL。IS NULL是一个整体，如果将IS换成“=”，将不能查询出任何结果，数据库系统会出现“空结果集”的提示。同理，IS NOT NULL中的IS NOT也不能换成“!=”或“<>”。

**语法规则如下：**

字段名称 IS [NOT] NULL

例如，查询注册会员表中email不为空，电话号码不为空，年龄大于30或小于20岁的男性会员的部分信息。

|  |
| --- |
| mysql> SELECT username, age , sex , tel , email FROM tb\_user where email IS NOT  NULL and tel IS NOT NULL and (age>30 or age<20) and sex=1;  +----------+-----+-----+-------------+-----------------------+  | username | age | sex | tel | email |  +----------+-----+-----+-------------+-----------------------+  | wangxiangjun | 32 | 1 | 13520551441 | wangxiangjun2008@gmail.com |  | wangzeqi | 18 | 1 | 13520551441 | wangzeqi@gmail.com |  +----------+-----+-----+-------------+-----------------------+  2 rows in set (0.00 sec) |

该示例中使用了逻辑运算符or和and。具体含义见下一节。

1. BETWEEN AND等同于a>=min and a<=max，判断某个字段的值是否在指定的范围内。如果在指定范围内，则满足查询条件，该记录被查询出来，否则不满足查询条件。BETWEEN AND和NOT BETWEEN AND 在查询指定范围的记录时很有用，常用于年龄段查询、考试分数查询、员工薪资水平查询等。

**语法规则如下：**

字段名称 [NOT ] BETWEEN 数值1 AND 数值2

例如，查询注册会员表中年龄在20岁到25岁之间（含20和25）的所有会员的部分信息。

|  |
| --- |
| mysql> SELECT username, age , sex , tel , email FROM tb\_user WHERE age BETWEEN  20 AND 25;  +---------------+-----+-----+-------------+---------------------------+  | username | age | sex | tel | email |  +---------------+-----+-----+-------------+---------------------------+  |liangyue | 22 | 0 | 13802812356 | liangyue@liangyue.com |  |shuainan | 21 | 1 | 13525566665 | shuainan@sohu.com |  |zhangyouqiang | 22 | 1 | 13465862586 | liqiang@sina.com |  +---------------+-----+-----+-------------+---------------------------+  3 rows in set (0.00 sec) |

1. IN运算符可以判断某个字段的值是否在指定的集合中。如果字段的值在集合中，则满足查询条件，该记录被查询出来。否则则不满足查询条件。

**语法规则如下：**

字段名称 [NOT ] IN （元素1 , 元素2 , 元素3 …）

备注：各个元素间用逗号隔开，字符型元素需要加单引号。

例如，查询注册会员表中ID分别是23，25的会员的信息。

|  |
| --- |
| mysql> SELECT id , username, age , sex , tel , email FROM tb\_user WHERE id IN (23,25);  +----+----------+-----+-----+-------------+---------------------------+  | id | username | age | sex | tel | email |  +----+----------+-----+-----+-------------+---------------------------+  | 23 | xiaozhuang | 20 | 1 | 15210974594 | xiaozhuang@xiaozhuang.com |  | 25 | zhangzhen | 21 | 1 | 13665544556 | zhangzhen @zhangzhen.com |  +----+----------+-----+-----+-------------+---------------------------+  2 rows in set (0.00 sec) |

例如，查询注册会员表中用户名不是“wangxiangjun”和“wangzeqi”的会员的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username NOT IN ('wangxiangjun' , 'wangzeqi'); |

备注：当IN中的记录为字符型元素时，要给元素加单引号。

1. LIKE是通配符匹配运算符，可以匹配字符串是否相等。如果字段的值与指定的字符串匹配，则满足查询条件，该记录被查询出来。如果与指定的字符串不匹配，则不满足查询条件。

**语法规则如下：**

字段名称 [NOT] LIKE ‘字符串’

**备注：**

‘字符串’表示指定用来匹配的字符串，该字符串必须加单引号或双引号。‘字符串’可以是一个完整的字符串，也可以是包含百分号“%”和下划线“\_”的通配字符。

“%”和 “\_”有很大的差别：

* + 1. “%”可以代表任意长度的字符串，长度可以是0。例如，b%k可以匹配book、break、bak、back等；
    2. 下划线“\_”只能表示单个字符。例如，b\_k可以匹配bak、bok、buk，但是却不能匹配book、break。

例如，查询注册会员表中姓王，且名字是三个字的会员的信息。

|  |
| --- |
| mysql> SELECT id , username , email FROM tb\_user WHERE username LIKE '王\_\_'; |

注意此SQL语句中的LIKE '王\_\_'，其中是两个下划线。而如果使用LIKE '王%'则查出来的数据要比前者多，因为‘%’匹配任意长度的字符串。如果要使用‘%’来实现名字三个字，可以借助上章中所学的MySQL常用函数length()来实现获取字符串的长度。

|  |
| --- |
| mysql> SELECT id , username , email FROM tb\_user WHERE username LIKE '王%' AND length(username)=6; |

因为username字段使用的是gbk编码，而gbk编码是双字节编码。因此三个中文字的长度应该是6。

1. regexp是正则表达式匹配运算符。正则表达式是用某种模式去匹配一类字符串的方式，其查询能力比通配符查询能力更强大，而且更灵活。正则表达式可以用于非常复杂的查询。

**语法规则如下：**

字段名称 regexp “匹配方式”

**备注：**

MySQL中的正则表达式与java、php、javascript中的基本一致。以下是常用的正则表达式的模式字符：

* + 1. ^ 在字符串开始处进行匹配
    2. $ 在字符串末尾处进行匹配
    3. . 匹配任意单个字符
    4. [] 匹配括号内任意字符
    5. [^ ] 匹配不在括号内的任意字符
    6. a\* 匹配0或多个a（包括空串）
    7. a+ 匹配1个或多个a（不包括空串）
    8. a？ 匹配0个或1个a
    9. a1|a2 匹配a1或a2
    10. a{m} 匹配m个a
    11. a{m,} 匹配至少m个a
    12. a{m,n} 匹配至少m个，至多n个a
    13. a{,n} 匹配0到n个a
    14. () 将模式元素组成单一元素

例1，查询会员注册表中用户名姓“王”的用户的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username REGEXP '^王'; |

上例中的正则表达式一定要使用“^ ”，表示在字符串开始处进行匹配。如果上面的例子使用通配符查询，也可以实现同样的效果。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username LIKE '王%'; |

例2，实际工作中，我们经常想不起来某个用户名的全部，只记得第一个字母是“s”，最后一个是“n”，中间的只隐约记得是四个字母。现在希望找到符合要求的用户名有哪些？

查询会员注册表中用户名首字母是“S”，最后一个字母是“n”，两个字母之间有四个字母，请查找出符合要求的所有的账户信息。

“^ ”，表示在字符串开始处进行匹配；

“$”，表示在字符串末尾处进行匹配；

“. ”，匹配任意单个字符。

于是例2的正则表达式可以写成’^S….n$’。一定要注意两个字母中间的那些点。每个点表示一个单字符。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username REGEXP 'S....n'; |

例3，对于例2稍加修改。如果想查会员注册表中用户名姓“王”，最后一个字是“军”，两字中间只有一个字，但是忘记了。使用上面的做法是否生效呢？

查询会员注册表中用户名姓“王”，姓名一个三个字，最后一个字是“军”的用户的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username REGEXP '^王.军$'; |

如果这么写，你会发觉无法查到想要的记录。为什么呢？原因是中文字符涉及到字符集问题，如果使用一个点，无法正确匹配一个中文字。那么我又不知道数据库设计的时候，该字段是GBK还是UTF8字符集，那么到底该写几个点呢？再者，写N个点很容易出错，于是另一个正则中的原子就会派上用场。

“a{m,n}”， 匹配至少m个，至多n个a。

因此例3的正则表达式可以写成“.{2,3}”。如果是GBK编码，则一个中文是2字符，如果是UTF8编码，则一个中文字是3字符。

因此，查询会员注册表中用户名姓“王”，姓名一个三个字，最后一个字是“军”的用户的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username REGEXP '^王.{2,3}军$'; |

例4，对于例3稍加修改。如果想查会员注册表中用户名姓“王”，最后一个字是“军”，两字中间只有一个字，记不太清楚了，好像是“军，照，西，雨，宇，忠”这些字中的一个。那么又该如何写正则呢？如果例3 的做法，查出来的一定是范围过宽，不够精确。于是“[]”派上了用场。

“[]” ，匹配括号内任意字符。

查询会员注册表中用户名姓“王”，姓名一个三个字，最后一个字是“军”，中间一个字是“向、像、项、湘、象、相、乡、翔”中的其中一个，请查出该用户的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE username REGEXP '^王[向像项湘象相乡翔]{2,3}军$'; |

备注：之所以要在“[]”后加{2,3}，也是考虑到一个中文字占2到3个字符。

例5，查询会员注册表中email是163邮箱的所有用户的信息。

|  |
| --- |
| mysql> SELECT \* FROM tb\_user WHERE email REGEXP '@163\.com$'; |

备注：因为点“.”在正则表达式中代表特殊的含义，因此email中的“.com”需要在点前增加转义符“\”。

### 逻辑运算符

逻辑运算符又称为布尔运算符，用来确认表达式的真和假。MySQL支持四种逻辑运算符，包括：NOT或! ，AND或&&，OR或||， XOR。作用分别是逻辑非、逻辑与、逻辑或、逻辑异或。在实际工作中常用的为前三个，XOR一般不使用，这里不做讲解。

例如，查询注册会员表中email不为空，电话号码不为空，年龄大于30或小于20岁的男性会员的信息。

|  |
| --- |
| mysql> SELECT username, age , sex , tel , email FROM tb\_user where email IS NOT  NULL and tel IS NOT NULL and (age>30 or age<20) and sex=1;  +----------+-----+-----+-------------+-----------------------+  | username | age | sex | tel | email |  +----------+-----+-----+-------------+-----------------------+  | wangxiangjun | 32 | 1 | 13520551441 | wangxiangjun2008@gmail.com |  | wangzeqi | 18 | 1 | 13520551441 | wangzeqi@gmail.com |  +----------+-----+-----+-------------+-----------------------+  2 rows in set (0.00 sec) |

## SQL高级查询

### 条件查询

实际工作中，不加条件的查询语句是很少的。用户几乎很少去查询所有的记录，而都是根据需求设置查询条件，由限定条件查询出符合要求的数据。实现方法是使用WHERE条件表达式。因为运算符是构成WHERE 表达式的必要成分，这也就是为什么必须要先学习MySQL运算符的原因了。

**语法规则如下：**

SELECT …… WHERE 条件表达式

WHERE条件表达式会充分运用各种比较运算符，而多个条件之间还可以使用or，and等逻辑运算符进行多条件的查询。设置的条件越多，查询语句的限制就会越多，能够满足所有条件的记录就会越少。

**示例代码详见上一节，“比较运算符”。**

### 查询不重复的记录

实际工作中，经常会需要将表中某字段的重复记录去掉后再显示出来，实现方法是使用distinct关键字。distinct关键字能消除相同的记录。“distinct”的意思是“有区别的，与其它不同的”。

**语法规则如下：**

SELECT DISTINCT 字段名称

例如，在新闻表中，有时候会重复插入相同标题、相同出处、相同新闻类型的信息，那么在显示的时候就需要去除这些重复的信息。

下面的例子是：显示新闻表中标题、新闻来源、新闻类型都不重复的所有信息。

|  |
| --- |
| mysql> SELECT distinct title , source , typeid FROM tb\_news;  +-----------------------------+----------+--------+  | title | source | typeid |  +-----------------------------+----------+--------+  |菲律宾示威者抗议日本猎杀海豚 |新华网 | 4 |  |全民抢入手!超级娱乐S60诺记N81近底线 |太平洋 | 5|  |俄罗斯承诺将援南奥塞梯每人2000美元 | sohu.com | 6 |  |菲律宾示威者抗议日本猎杀海豚 |新华网 | 2 |  +-----------------------------+----------+--------+  4 rows in set (0.03 sec) |

**【备注：】**上面的例子中，虽然有两条的标题和新闻来源都一样，但是因为新闻类型不同，所以两条都显示出来了，如果连新闻类型，也就是typeid也一样，那么只显示其中一条了。

### 排序和限制查询

从表中查询出来的记录可能是无序的，或者排列顺序不是用户所期望的。为了使得查询结果的顺序满足用户要求，可以使用ORDER BY来对记录进行排序。

**语法规则如下：**

SELECT … ORDER BY 字段名称 [ASC | DESC]

ASC表示升序排列，DESC表示降序排列。默认情况下，按照ASC方式排序。ORDER BY后面可以跟多个不同的排序字段，并且每个排序字段可以有不同的排序方式。

**语法规则如下：**

SELECT \* FROM TABLE\_NAME [WHERE ...] [ORDER BY field1 [DESC|ASC] ,

field2 [DESC|ASC]...];

对于排序后的记录，如果希望只显示一部分，而不是全部，那么可以使用LIMIT关键字来实现。

**LIMIT语法规则如下：**

SELECT \* FROM TABLE\_NAME [WHERE ...] [ORDER BY ...] [LIMIT OFFSET , ROW\_COUNT];

LIMIT关键字有两种使用方式：

1. 不指定OFFSET偏移量，用法为：LIMIT 记录数；
2. 指定OFFSET偏移量，用法为： LIMIT 偏移量 , 记录数。

LIMIT和ORDER BY关键字联合使用，解决了页面数据分页展示问题。

例如，查询会员注册表中，最后注册的5名会员的部分信息。

|  |
| --- |
| mysql> SELECT id , username , age , email FROM tb\_user ORDER BY id DESC LIMIT 0,5;  +----+---------------+-----+----------------------------+  | id | username | age | email |  +----+---------------+-----+----------------------------+  | 42 | Steven | 17 | 14455588@qq.com |  | 41 | zhangyouqiang | 22 | liqiang@sina.com |  | 39 | xiangjunwang | 32 | wangxiangjun2008@gmail.com |  | 38 | zeqiwang | 18 | zeqi@qq.com |  | 37 | shaoqi | 30 | shaoqi@163.com |  +----+---------------+-----+----------------------------+  5 rows in set (0.00 sec) |

### 分组聚合查询

实际工作中，经常会需要对表中数据按组统计总数量，按照组查其中的最大、最小或平均值，按照组求其总和等等操作。GROUP BY 关键字可以将查询结果按照某个字段或多个字段进行分组。

**GROUP BY语法规则如下：**

SELECT \* FROM TABLE\_NAME [WHERE ...] [GROUP BY 字段名] [HAVING 表达式] [WITH ROLLUP];

1. HAVING关键字表示要对分类后的结果再进行条件过滤；
2. WITH ROLLUP是可选语法，表明是否对分类聚合后的结果进行再汇总。如果使用该关键字，则会在所有记录的最后增加一条记录，该记录是上面所有记录的总和。

GROUP BY可以单独使用，但是单独使用时，只能查询出每个分组的一条记录。这样使用的意义不大，因此，一般在使用聚合函数的时候才使用GROUP BY关键字。

常用聚合函数有：COUNT()、SUM() 、MAX()、MIN()、AVG()。

1. COUNT()用来统计记录的总条数；
2. SUM()用来计算某个字段的符合条件的所有值的总和；
3. MAX()用来查询某字段的符合条件的所有值中的最大值；
4. MIN()用来查询某字段的符合条件的所有值中的最小值；
5. AVG()用来计算某字段的符合条件的所有值的平均值。

**运用聚合函数的语法如下：**

SELECT [field1, field2 , field3...] FUNCTION\_NAME

FROM TABLE\_NAME

[WHERE where\_condition]

[GROUP BY field1 , field2...]

[HAVING where\_condition]

[WITH ROLLUP]

聚合函数一般情况下都跟GROUP BY一起使用。在MYSQL5.1官方手册中就将聚合函数称为GROUP BY函数。因为聚合函数通常都是用来计算某一组数据的总和、最大值、最小值、平均值。

**【备注：】**使用聚合函数的时候，不一定非要跟GROUP BY 关键字一起使用。例如统计表中所有的记录数时，就可以直接使用COUNT()，统计某个表中所有记录中的最大值、最小值、平均值和总和时就不用GROUP BY关键字。但是反之，GROUP BY 关键字一般都在使用聚合函数的时候才更有意义。

例1，查询用户注册表中男性和女性各有多少注册用户。

|  |
| --- |
| mysql> SELECT sex , COUNT(\*) FROM tb\_user GROUP BY sex;  +-----+----------+  | sex | COUNT(\*) |  +-----+----------+  | 0 | 1 |  | 1 | 14 |  +-----+----------+  2 rows in set (0.00 sec) |

例1中，性别为0（女）的有1位，性别为1（男）的有14位。

从例1中看到查询出来的字段有sex和count(\*) ，count(\*)看起来不够直观，最后用一个更加直观的名字来表示这个列。为了解决这个问题，可以为字段取别名。

**MySQL中为字段取别名的语法规则如下：**

字段 [AS] 别名

因此例1可更改为：

查询用户注册表中男性和女性各有多少注册用户。

|  |
| --- |
| mysql> SELECT sex AS 性别 , COUNT(\*) AS 总数 FROM tb\_user GROUP BY sex;  +-----+----------+  | 性别 | 总数 |  +-----+----------+  | 0 | 1 |  | 1 | 14 |  +-----+----------+  2 rows in set (0.00 sec) |

例2，查询用户注册表中男性和女性各有多少注册用户，以及注册会员的总数。

分析：只需对例1稍加修改，增加上WITH ROLLUP即可。

|  |
| --- |
| mysql> SELECT sex AS 性别 , COUNT(\*) AS 总数 FROM tb\_user GROUP BY sex WITH ROLLUP;  +-----+----------+  | 性别 | 总数 |  +-----+----------+  | 0 | 1 |  | 1 | 14 |  | NULL | 15 |  +-----+----------+  3 rows in set (0.00 sec) |

**【备注：】**注意WITH ROLLUP的用法，在所有记录的最后加上了一条记录，该记录是上面所有记录的总和。

例3，查询会员注册表中男性和女性用户的平均年龄。

|  |
| --- |
| mysql> SELECT sex AS 性别 , AVG(age) AS 平均年龄 FROM tb\_user GROUP BY sex;  +-----+----------+  | 性别 | 平均年龄 |  +-----+----------+  | 0 | 22.0000 |  | 1 | 28.5000 |  +-----+----------+  2 rows in set (0.03 sec) |

例4，查询会员注册表中男性和女性会员的最大年龄。

|  |
| --- |
| mysql> SELECT sex AS 性别 , MAX(age) AS 最大年龄 FROM tb\_user GROUP BY sex ;  +-----+----------+  | 性别 | 最大年龄 |  +-----+----------+  | 0 | 22 |  | 1 | 32 |  +-----+----------+  2 rows in set (0.00 sec) |

例5，以新闻表为例。新闻有各种新闻类型，每种新闻类型下都有很多新闻信息。

统计新闻信息表中每种新闻类型各有多少条新闻。

|  |
| --- |
| mysql> SELECT typeid AS新闻类型, COUNT(\*) AS 新闻总条数 FROM tb\_news GROUP BY typeid;  +--------+----------+  | 新闻类型 | 新闻总条数 |  +--------+----------+  | 1 | 50 |  | 2 | 21|  | 3 | 2 |  | 4 | 15 |  | 5 | 12 |  +--------+----------+  5 rows in set (0.00 sec) |

例6，以订单明细表为例。

通过订单明细表，统计每种商品售出的总数。（只要下订单即可，不论是否发货或到货）

分析：商品明细表中有字段是goods\_id，这个是商品的唯一标识id，有一个字段是goods\_count，也就是每个订单中购买该商品的数量。那么只要按照goods\_id分组，然后统计goods\_count的总和，那么就可以达到查询目的。

|  |
| --- |
| mysql> SELECT goods\_id AS 商品id , SUM(goods\_count) AS 销售总量 FROM tb\_orderlist GROUP BY goods\_id;  +----------+------------------+  |商品id |销售总量|  +----------+------------------+  | 2 | 7 |  | 3 | 2 |  | 4 | 4 |  +----------+------------------+  3 rows in set (0.00 sec) |

例7，对例6稍加修改。统计新闻信息表中新闻类型的总条数大于20条的新闻类型。

|  |
| --- |
| mysql> SELECT typeid AS 新闻类型, COUNT(\*) AS 新闻总条数 FROM tb\_news GROUP BY typeid HAVING 新闻总条数>20;  +--------+----------+  | 新闻类型 | 新闻总条数 |  +--------+----------+  | 1 | 50 |  | 2 | 21|  +--------+----------+  2 rows in set (0.00 sec) |

备注：本例中查询出的记录是在例5的基础上再次进行了一次过滤，也就是将新闻总条数中小于20的都过滤掉了。从本例可以看出，HAVING是对聚合后的结果再进行条件的过滤。那么HAVING 和WHERE区别在哪里呢？

**【备注：】**HAVING和WHERE的区别：

HAVING是对聚合后的结果再进行条件的过滤，而WHERE是在聚合前就对记录进行过滤，如果逻辑允许，我们尽可能用WHERE先过滤记录，这样因为结果集减小，将对聚合的效率大大提高，最后再根据逻辑判断是否还需要用HAVING进行再次过滤。

### 多表连接查询

实际工作中，有时候需要查询的信息在多个表中，仅仅通过查询一个表，无法一次性获取到所有需要的信息。于是就需要使用多表的连接查询。

当多个表中存在相同意义的字段时，就可以通过该字段来连接这几个表。连接查询就是将两个或两个以上的表按某个条件连接起来，从中选取需要的数据。连接查询是同时查询两个或两个以上的表的时候使用的。

连接查询又分为内连接和外连接。后者又分为左连接和右连接。而内连接时最常用的连接查询。

1. 内连接： 内连接仅选出两张表中互相匹配的记录，而外连接可以选出其它不匹配的记录。

**内连接语法规则如下：**

SELECT 字段1 , 字段2 … FROM 表名1 , 表名2 …

WHERE 表名1.字段 = 表名2.字段;

**内连接语法规则2（该语法规则是根据外连接语法规则来的）：**

SELECT 字段1 , 字段2 … FROM 表名1

**INNER JOIN** 表名2

ON 表名1.字段 = 表名2.字段;

为了能一次性查出多个表的信息出来，就要将多个表连接起来，而多个表能连接起来的关键就在于这些表中有表示相同意义的字段。找到这个字段，将其作为WHERE查询条件，就可以实现内连接查询。

例如查询新闻表，可以查询到新闻id、新闻标题、新闻来源、新闻类型id、新闻内容等等。虽然能查出新闻的类型id，但是这个类型id代表的是什么新闻，却不能一下子知道，只能再去查新闻类型表，才能知道每个类型id代表的中文含义。那么能不能就查询一次，一下子就能得到新闻id、新闻标题、新闻来源、新闻类型的名称、新闻内容这些信息呢？

能！这就要通过表连接查询，把新闻表和新闻类型表连接起来。那么连接的关键是什么呢？新闻表（tb\_news）中有个typeid字段，代表新闻的类型id，这个值正好是新闻类型表(tb\_newstype)中的id值 ， 也就是“ tb\_news.typeid=tb\_newstype.id ”。于是连接点就找到了。

例1，查询新闻表中最新发布的一条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT tb\_news.id , title , source , typename FROM tb\_news , tb\_newst  ype WHERE tb\_news.typeid=tb\_newstype.id LIMIT 1;  +----+----------------+---------------+----------+  | id | title | source | typename |  +----+----------------+---------------+----------+  | 56 | 菲律宾示威者抗议日本猎杀海豚 | 新华网 | 国内新闻 |  +----+----------------+---------------+----------+  1 row in set (0.00 sec) |

从例1中看到，查询新闻id的时候要写成tb\_news.id。因为将两个表连接起来查询时，两个表中都有id字段，那么到底要显示那个表的id呢，就需要在字段前写上“表名称.”。如果字段比较多，或者表名称比较长，那么在查询中直接使用表名很不方便，这个时候可以为表取一个别名。

**MySQL中为表取别名的语法规则如下：**

表名 表的别名

因此例1可更改为：

查询新闻表中最新发布的一条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT n.id , title , source , typename FROM tb\_news n , tb\_newstype t WHERE n.typeid=t.id LIMIT 1;  +----+----------------+---------------+----------+  | id | title | source | typename |  +----+----------------+---------------+----------+  | 56 | 菲律宾示威者抗议日本猎杀海豚 | 新华网 | 国内新闻 |  +----+----------------+---------------+----------+  1 row in set (0.00 sec) |

通过修改后例1可以看到，新闻表tb\_news 的别名是n ，新闻类型表tb\_newstype 的别名是t 。 有了表别名，那么在写WHERE条件的时候，只用写“表别名1.字段=表别名2.字段”。而且展示字段列表的时候，只用写“表别名.字段”即可。

例2，对例1稍加改动，增加一个查询条件。

查询新闻表中新闻类型为3的，最新发布的一条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT n.id , title , source , typename ,typeid FROM tb\_news n , tb\_newstype t WHERE n.typeid=t.id AND n.typeid=3 LIMIT 1;  +----+----------------+---------------+----------+  | id | title | source | typename |  +----+----------------+---------------+----------+  | 56 | 菲律宾示威者抗议日本猎杀海豚 | 新华网 | 国内新闻 |  +----+----------------+---------------+----------+  1 row in set (0.00 sec) |

例2是让大家明白，如果在内连接查询后，还有其它的查询条件，只要在WHERE子句中增加AND表达式即可。

例3，用内连接查询的另一种写法实现。

查询新闻表中新闻类型为3的，最新发布的一条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT n.id , title , source , typename FROM tb\_news n INNER JOIN tb\_newstype t ON n.typeid=t.id WHERE n.typeid=3 LIMIT 1;  +----+----------------+---------------+----------+  | id | title | source | typename |  +----+----------------+---------------+----------+  | 56 | 菲律宾示威者抗议日本猎杀海豚 | 新华网 | 国内新闻 |  +----+----------------+---------------+----------+  1 row in set (0.00 sec) |

1. 外连接：外连接分左连接和右连接。
   1. 左连接：包含所有的左边表中的记录，甚至是右边表中没有和它匹配的记录。
   2. 右连接：包含所有的右边表中的记录，甚至是左边表中没有和它匹配的记录。

**外连接语法规则如下：**

SELECT 字段1 , 字段2 … FROM 表名1

**LEFT JOIN** | **RIGHT JOIN** 表名2

ON 表名1.字段 = 表名2.字段;

LEFT JOIN表示左连接，RIGHT JOIN表示右连接，ON后面接的是连接条件。

通过左连接查询，可以查询出“表1”中所有记录，而“表2”中只能查出跟“表1”匹配的记录，其它的记录无法显示。

通过右连接查询，可以查询出“表2”中所有记录，而“表1”中只能查出跟“表2”匹配的记录，其它的记录无法显示。

例4，用左连接查询实现。

查询新闻表中最新发布的5条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT n.id , title , source , typename FROM tb\_news n LEFT JOIN  tb\_newstype t ON n.typeid=t.id LIMIT 5;  +----+-----------------------------+---------------+----------+  | id | title | source | typename |  +----+-----------------------------+---------------+----------+  | 56 |菲律宾示威者抗议日本猎杀海豚 |新华网 | 国内新闻 |  | 58 |超强S60智能怪物LG KT610评测|太平洋 | 手机新闻 |  | 59 |全民抢入手!超级娱乐S60诺记N81近底线 |太平洋 |手机新闻 |  | 61 |俄罗斯承诺将援南奥塞梯每人2000美元 | sohu.com |国际新闻 |  | 62 |美国宣布向格鲁吉亚提供10亿美元援助 | sina.com | NULL |  +----+-----------------------------+---------------+----------+  5 rows in set (0.00 sec) |

使用左连接查询后，出现一条新闻的新闻类型为NULL。这是因为新闻表中id为62的新闻的typeid为0，而在新闻类型表中没有id为0的数据。这个时候两个表中有不匹配的记录。因为左连接查询，可以查询出“表1”（例3中就是新闻表）中所有记录，所以id为62的记录虽然在新闻类型表中无匹配记录，也被显示出来了。一定要注意跟内连接区别。

例5，用右连接查询实现。

查询新闻表中最新发布的5条新闻，要求有新闻id、新闻标题、新闻来源、新闻类型名称。

|  |
| --- |
| mysql> SELECT n.id , title , source , typename FROM tb\_news n RIGHT JOIN  tb\_newstype t ON n.typeid=t.id limit 5;  +------+-----------------------------+---------------+----------+  | id | title | source | typename |  +------+-----------------------------+---------------+----------+  | 56 |菲律宾示威者抗议日本猎杀海豚 |新华网 | 国内新闻 |  | 58 |超强S60智能怪物LG KT610评测|太平洋 | 手机新闻 |  | 59 |全民抢入手!超级娱乐S60诺记N81近底线 |太平洋 |手机新闻 |  | 61 |俄罗斯承诺将援南奥塞梯每人2000美元 | sohu.com |国际新闻 |  | NULL | NULL | NULL |国内娱乐 |  +------+-----------------------------+---------------+----------+  5 rows in set (0.00 sec) |

使用右连接查询后，出现一条新闻信息为NULL， 而新闻类型为“国内娱乐”的记录。而此时新闻表中id为62的新闻也不显示了。因为右连接查询，可以查询出“表2”（本例中就是新闻类型表）中所有记录， 因此虽然暂时没有“国内娱乐”类型的新闻，但是该记录也被显示出来了。 此外，因为“表1”中只显示跟“表2”匹配的记录，因为id为62的记录跟“表2”不匹配，所以不再显示。一定要注意跟内连接区别。

### 子查询（嵌套查询）

实际工作中，需要将一个表的结果显查询出来，然后作为另一个SELECT语句的条件。这就要用到子查询。子查询就是将一个查询语句嵌套在另一个查询语句中。内层查询语句的查询结果，可以作为外层查询语句的条件。

子查询，也叫做内查询。虽然有些参考书中，将子查询和嵌套查询作为两种查询方式来讲，但是其实嵌套查询就是子查询。所以子查询，又叫做内查询或者嵌套查询。

通过子查询，可以实现多表之间的查询。

子查询中可以包含比较运算符：= 、!= 、< 、 > 、 <= 、 > 、>= 、IN 、 NOT IN ， 还可以包括：ANY 、SOME 、ALL 、 EXISTS 、NOT EXISTS等关键字。

子查询和表连接查询都可以实现多表之间的查询。某些情况下，子查询可以转化为表连接查询。但是在查询效率上会有所不同，需要大家逐步积累和总结。

（在mysql4.1版本前不支持子查询，需要用表连接查询来实现子查询的功能。表连接很多情况下用于优化子查询。

子查询可总结为以下七类：

* + 利用内层查询语句派生出新表的子查询；
  + 利用内层查询语句作为表达式的子查询；
  + 带常规比较运算符的子查询。包括的运算符有：= 、<>或!= 、< 、 > 、 <= 、 > 、>=等 ；
  + 带IN关键字的子查询；
  + 带EXISTS关键字的子查询；
  + 带ANY或SOME关键字的子查询；
  + 带ALL关键字的子查询。

1. 利用内层查询语句派生出新表的子查询。

**语法结构如下：**

SELECT 字段1 , 字段2 … FROM （SELECT子查询语句） [AS] 别名;

从语法规则上可以看出，平时FROM后跟的是表名，而此时后面跟另一个SELECT子查询语句。这里将内层查询语句派生出一个新表，然后给这个派生出的新表起个别名。

例1，统计新闻信息表中新闻类型的总条数大于20条的新闻类型。

细心的同学会发觉，这个例子就是“4.2.4分组聚合查询”中的例7。那么这个例子又跟子查询有什么关系呢？好，我们先回顾一下4.2.4分组聚合查询例7的SQL语句的写法：

|  |
| --- |
| mysql> SELECT typeid AS 新闻类型, COUNT(\*) AS 新闻总条数 FROM tb\_news GROUP BY typeid HAVING 新闻总条数>20;  +--------+----------+  | 新闻类型 | 新闻总条数 |  +--------+----------+  | 1 | 50 |  | 2 | 21|  +--------+----------+  2 rows in set (0.00 sec) |

大家能看到，显示出两列，“新闻类型”和“新闻总条数”。如果我只想要得到符合条件的“新闻类型”，不要显示“新闻总条数”可以做到呢？第二列是由聚合函数COUNT(\*)产生的，如果不要第二列，就不可能有语句后面的HAVING条件。到底如何做呢？利用子查询就可以做到。

例2，统计新闻信息表中新闻类型的总条数大于20条的新闻类型的id。仅仅显示新闻类型id一列，不要显示新闻条数。

|  |
| --- |
| mysql> SELECT typeid FROM (select typeid , COUNT(\*) FROM tb\_news GROUP BY typeid HAVING count(\*) >20 ) AS a;  +--------+  | typeid |  +--------+  | 1 |  | 2 |  +--------+  2 rows in set (0.00 sec) |

大家能看到例2跟例1结果上的差异，一个有两个字段，一个只显示一个字段。这就是利用内层查询语句派生出新表的子查询的效果。例2的SQL语句中记得给派生出的新表起别名，其中AS关键字可以使用，也可以不用。

1. 利用内层查询语句作为表达式的子查询。

在“4.2.4分组聚合查询” 的例3中有这样的查询例子。查询会员注册表中男性和女性用户的平均年龄。

|  |
| --- |
| mysql> SELECT sex AS 性别 , AVG(age) AS 平均年龄 FROM tb\_user GROUP BY sex;  +-----+----------+  | 性别 | 平均年龄 |  +-----+----------+  | 0 | 22.0000 |  | 1 | 28.5000 |  +-----+----------+  2 rows in set (0.00 sec) |

查询结果是如上，两行数据，两列字段。这样的显示结果不够直观，能不能显示的更加直观些？如果查询要求是就显示成一行数据，两列字段，该如何实现呢？接下来的例3就是利用内层查询语句作为表达式的方式来实现的。

例3，查询会员注册表中男性和女性用户的平均年龄。要求显示成一行两列。

|  |
| --- |
| mysql> SELECT (SELECT AVG(age) FROM tb\_user WHERE sex=1) AS 男会员平均年龄 , (SELECT AVG(age) FROM tb\_user WHERE sex=0) AS 女会员平均年龄 ;  +-----+----------+  |男会员平均年龄| 女会员平均年龄 |  +-----+----------+  | 28.5000 | 22.0000 |  +-----+----------+  2 rows in set (0.00 sec) |

1. 带常规比较运算符的子查询。

包括的运算符有：= 、<>或!= 、< 、 > 、 <= 、 > 、>=等。

例4 ， 查询注册会员表中年龄最大的男性会员的注册信息。

|  |
| --- |
| mysql> SELECT id , username , age , sex FROM tb\_user WHERE age = (SELECT MAX(age) FROM tb\_user WHERE sex=1);  +----+----------+-----+-----+  | id | username | age | sex |  +----+----------+-----+-----+  | 1 | 王向军 | 32 | 1 |  +----+----------+-----+-----+  1 row in set (0.00 sec) |

例4中的子查询语句返回结果只有一个值，所以可以使用“age=”直接赋值，如果子查询语句的返回结果是一组记录，又该如何呢？答案是可以借助带IN关键字的子查询。

1. 带IN关键字的子查询。

稍微修改例2，将查询条件更改为：

例5，查询新闻总数大于100条的那几个类型的新闻，只需要显示最后发布的5条即可。

分析：新闻总数大于100条的那几个新闻类型有哪些呢？通过例2就可以获取到他们的typeid，那么在新闻表中查找typeid是这些的新闻不就可以了吗？

|  |
| --- |
| mysql> SELECT \* FROM tb\_news WHERE typeid IN  (SELECT typeid FROM (SELECT typeid , COUNT(\*) FROM tb\_news GROUP BY typeid HAVING COUNT(\*) >0 ) AS a); |

例5中使用了关键字IN。因为其子查询语句返回的不是一个typeid，而是一组typeid。所以不能使用“=”，就只能使用IN关键字。

1. 带EXISTS关键字的子查询。

EXISTS表示存在，使用EXISTS关键字时，内层查询语句不返回查询记录。只返回true或者false。当返回true，外层查询语句将进行查询，当返回false的时候，外层查询语句不查询或查询不出任何记录。

以新闻表表和新闻类型表为例。新闻类型表中有id为1、2、3、4、5这5种新闻类型，相应在新闻表中也会存在typeid为这5种的新闻。某一日，用户将新闻类型表中的id为5的记录删除了，而新闻表中typeid为5的数据未同时删除。那么新闻表中的typeid为5的数据就“无家可归”，因此在显示在网页上的时候也不应该显示。可是typeid为5的记录明明又在新闻表中存在着，如何让它不显示呢？

为了解决以上的问题，可以用表连接查询中的内连接。因为内连接是两个表中的记录都完全匹配的时候才能查询出来。除了内连接外，还有一种做法就是利用带EXISTS关键字的子查询。

**EXISTS子查询语法结构如下：**

SELECT 字段1 , 字段2 … FROM 表名

WHERE [ where条件语句 ] [ AND ]

EXISTS （SELECT子查询语句）;

如果查询的时候外层查询语句没有额外的where条件，那么WHERE 后就直接连EXISTS，否则就需要写where条件语句和AND关键字，再连上EXISTS。

例6，查询新闻表中新闻类型id为5的新闻信息，要求是类型id必须在新闻类型表中真实存在，如果不存在，就不要查询出任何记录。

|  |
| --- |
| mysql> SELECT \* FROM tb\_news WHERE typeid=5 AND EXISTS (SELECT id FROM tb\_newstype WHERE id=5);  Empty set (0.00 sec) |

例6中，因为新闻类型表中id为5的记录已经被删除了。所以查询结果为空。

某些情况下，子查询可以转化为表连接查询来实现。例6即是如此。转换为内连接后的SQL语句如下：

|  |
| --- |
| mysql> SELECT \* FROM tb\_news n , tb\_newstype t WHERE n.typeid = t.id AND n.typeid=5 ;  Empty set (0.00 sec) |

**【备注：】**EXISTS关键字与前面几种子查询语句都不一样。使用EXISTS关键字时，内层查询语句只返回true和false。而前面几种子查询，其内层查询语句都会返回查询到的记录。

1. 带ANY或SOME关键字的子查询和带ALL关键字的子查询。

实际工作中，使用ANY 、SOME 、 ALL的情形比较少。所以仅仅作为了解，知道子查询中可以有这三个关键字即可。

### 记录联合查询

某些情况下，需要将几个SELECT语句查询出来的结果合并起来显示。这个时候需要用表联合来实现。进行合并操作使用UNION和UNION ALL关键字来实现。

**记录联合语法规则如下：**

SELECT 语句1

**UNION** | **UNION ALL**

SELECT 语句2

**UNION** | **UNION ALL**

SELECT 语句3

**UNION** | **UNION ALL** …

SELECT 语句n;

例如：从注册会员表中查出年龄最大和最小的所有会员的信息。

|  |
| --- |
| mysql> SELECT id , username , age , sex FROM tb\_user WHERE age = (SELECT MAX(age) FROM tb\_user )  UNION  SELECT id , username , age , sex FROM tb\_user WHERE age = (SELECT MIN(age) FROM tb\_user );  +----+----------+-----+-----+  | id | username | age | sex |  +----+----------+-----+-----+  | 1 | 王向军 | 32 | 1 |  | 28 |佟新华 | 17 | 1 |  | 42 | Steven | 17 | 1 |  +----+----------+-----+-----+  3 rows in set (0.00 sec) |

如果不使用记录联合，那么一次性获得的要么是年龄最大的会员的信息，要么就是年龄最小的会员的信息。而使用UNION后，这些信息都可以同时显示出来。

**【备注：】**UNION和UNION ALL的区别：

UNION ALL是把结果集直接合并在一起，而UNION是将UNION ALL后的结果进行一次DISTINCT处理，消除其中所有的重复记录。

## 约束(constraint)

当创建数据表的时候，为某一个字段设置约束，表示限定该列的数值内容。也可以通过修改表结构添加约束。

### 非空约束：not null

1、如果某个字段添加了非空约束，那么该字段的数值不能为null，对于mysql而言，表示什么都没有，未知的。不是0，也不是空""，更不是"NULL"。

create table stu (

sid int ,

sname varchar(20) not null, # 定义sname字段值不能为空

age int,

gender varchar(10)

);

2、删除非空约束：

alter table stu modify sname varchar(20);

3、添加非空约束：

alter table stu modify sname varchar(20) not null;

该列已有的数据不能有null数值。

### 唯一性约束：unique

1、唯一性约束强调所在的字段所有的数值都是唯一的。不允许重复。允许空值（多个）。

create table stu (

sid int primary key auto\_increment,

sname varchar(20) unique, # 定义sname字段值必须唯一

age int,

gender varchar(10)

);

或者,使用constraint 约束名 unique 字段。

create table stu (

sid int auto\_increment,

sname varchar(20),

age int,

gender varchar(10),

[constraint ] unique (sname) # 指明name字段唯一

);

2、删除唯一约束：

alter table stu modify sname varchar(20);

3、添加唯一约束：

alter table stu modify sname varchar(20) not null;

该列已有的数据不能有null数值。

4、为已有的字段设置约束

alter table 表名 modify 字段 约束

5、添加新的字段，带约束

alter table 表名 add 字段 数据类型 约束

6、先将已有的字段删除，重新添加带约束的。

alter table 表名 drop column 列名 alter table 表名 add 列名 数据类型 约束

### 主键约束：primary key。

1、主键约束：要求该字段数值不允许空，而且数值唯一。(not null+unique)

主键约束用于唯一的标识表中的每一个行记录的。一个表中最多只能有一个主键约束。

create table stu (

sid int primary key auto\_increment, # 定义sid作为stu表的主键

sname varchar(20),

age int,

gender varchar(10)

);

或者,使用constraint 约束名 primary key 字段。

create table stu (

sid int auto\_increment,

sname varchar(20),

age int,

gender varchar(10),

[constraint [sid\_pk]] primary key (sid) # 指明sid作为stu表的主键

);

2、删除主键：

alter table stu drop primary key;

3、添加主键：

alter table teachers add primary key(id);

4、主键自增： auto\_increment

要求该字段数值不允许空，而且数值唯一。所以我们通常会指定主键类型为整型，然后设置其自动增长，这样可以保证在插入数据时主键列的唯一和非空特性。

5、修改表时设置主键自动增长：

alter table stu modify sid int auto\_increment;

6、修改表时删除主键自动增长：

alter table stu modify sid int;

### 外键约束：foreign key，

1、外键

● 思考：怎么保证关系列数据的有效性呢？任何整数都可以吗？比如一个班级表中有班级编号，班级名称，班级位置。还有一个学生表，有学生编号，学生姓名，学生 性别，以及班级编号等。

● 答：必须是学生表中班级编号列存在的数据，可以通过外键约束进行数据的有效性验证

外键作用数据完整性。

班级表可以叫做父表(主表)：class，字段classno，该表的主键

学生表可以叫做子表(从表)：student，字段sid是该表的主键，classno是该表的外键

父表：department，字段deptno，该表的主键 子表：employee，字段empno，该表的主键，deptno，该表的外键

子表的外键是父表的主键。

● 当对于子表插入数据，外键字段的数值不能随便插入，而取决于父表的主键数值。

● 对于父表的删除操作，要看子表中是否有记录引用该数值。

2、外键的设置

（1）、方法一：在创建子表的时候，直接标明外键。

设置外键语法：

references 父表(主键);

constraint classno\_FK foreign key (classno) references class(classno);

创建数据表

create table scores(

sid int(4) primary key auto\_increment,

sname varchar(20),

sage int(3),

classno int(4),

constraint stu\_classno foreign key(classno) references class(classno) on delete cascade,

);

（2）、方法二：使用alter语句今天添加外键

alter table student add constraint stu\_classno foreign key(classno) references class(classno);

alter table student add constraint stu\_classno foreign key(classno) references class(classno) on delete cascade;

● 此时插入或者修改数据时，如果classno的值在student表中不存在则会报错

删除外键：

alter table student drop foreign key stu\_classno;

3、外键的级联操作

● 在删除class表的数据时，如果这个classno值在student中已经存在，则会抛异常

● 推荐使用逻辑删除，还可以解决这个问题

● 可以创建表时指定级联操作，也可以在创建表后再修改外键的级联操作

语法：

on delete cascade,删除父表的时候，子表的数据直接删除，级联删除

on delete set null，删除父表的时候，子表的数据置null。

● 级联操作的类型包括：

○ restrict（限制）：默认值，抛异常

○ cascade（级联）：如果主表的记录删掉，则从表中相关联的记录都将被删除

○ set null：将外键设置为空

○ no action：什么都不做

## 数据库的其他内容

### 事务

1、什么是事务

● 当一个业务逻辑需要多个sql完成时，如果其中某条sql语句出错，则希望整个操作都退回

● 使用事务可以完成退回的功能，保证业务逻辑的正确性

● 事务四大特性(简称ACID)：面试

○ 原子性(Atomicity)：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行

○ 一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致

○ 隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。不同的事务之间应该隔离开，使每个并发中的事务不会互相干扰

○ 持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障

● 要求：表的类型必须是innodb或bdb类型，才可以对此表使用事务

● 事务语句

开启事务：

begin

或者START TRANSACTION

束事务：

提交commit;

回滚rollback;

2、示例1

● 步骤1：打开两个终端，连接mysql，使用同一个数据库，操作同一张表

终端1：

select \* from students;

------------------------

终端2：

begin;

insert into students(sname) values('张飞');

● 步骤2

终端1：

select \* from students;

● 步骤3

终端2：

commit;

------------------------

终端1：

select \* from students;

3、示例2

● 步骤1：打开两个终端，连接mysql，使用同一个数据库，操作同一张表

终端1：

select \* from students;

------------------------

终端2：

begin;

insert into students(sname) values('张飞');

● 步骤2

终端1：

select \* from students;

● 步骤3

终端2：

rollback;

------------------------

终端1：

select \* from students;

当使用事务的情况：当数据被更改时：insert，update，delete。

### 视图

1、什么是视图

● 视图view，称作是虚拟表。(和真正的数据表非常像)。

● 视图本身是不包含数据。 视图的创建需要使用对某一个或多个数据表的查询(简单，复杂的)。如果对视图进行增删改操作，其实改的视图的基表。如果基表的数据改变了，视图中的数据也随之改变。

● 视图就理解为对一个查询语句的封装。

● 通俗的讲，视图就是一条SELECT语句执行后返回的结果集。所以我们在创建视图的时候，主要的工作就落在创建这条SQL查询语句上。

● 视图是对若干张基本表的引用，一张虚表，查询语句执行的结果，不存储具体的数据（基本表数据发生了改变，视图也会跟着改变）；可以跟基本表一样，进行增删改查操作(ps:增删改操作有条件限制)；

2、创建视图：

create [or repalce] view view\_name as 子查询 [with check option]视图的检查，通过视图的操作，要在视图的可见范围内。

3、示例：

create view stuscore as

select students.\*,scores.score from scores

inner join students on scores.stuid=students.id;

● 视图的用途就是查询

select \* from stuscore;

### 索引

1、为什么需要索引？

● 思考：在图书馆中，如何找到一本书呢？index

● 一般的应用系统，读写比例在10:1左右，而且插入操作和更新操作很少出现性能问题，遇到最多的，也是最容易出问题的，还是一些复杂的查询操作，所以查询语句的优化显然是重中之重。

● 当数据库中数据量很大时，查找数据会变得很慢。索引能够提高数据访问性能 ,快速的找到数据。

2、主键和唯一索引，都是索引，可以提高查询速度

1.索引可以快速的找到数据

2.数据默认会按照某种索引存储，一般是主键索引

3.索引的创建会增加物理上的开销

3、选择列的数据类型

越小的数据类型通常更好：越小的数据类型通常在磁盘、内存和cpu缓存中都需要更少的空间，处理起来更快

简单的数据类型更好：整型数据比起字符串，处理开销更小，因为字符串的比较更复杂

尽量避免NULL：该应指定列为NOT NULL，除非你想存储NULL。在MySQL中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。应该使用0，一个特殊的值或者一个空串代替空值。

操作

索引分为单列索引和组合索引

● 单列索引，即一个索引只包含单个列，一个表可以由多个单列索引，但这不是组合素银

● 组合索引，即一个索引包含多个列

4、查看索引

show index from table\_name;

5、创建索引

create index index\_name on table\_name(列)

6、删除索引

删除索引

drop index index\_name on table\_name

7、缺点

● 虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行insert，update和delete。因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件

● 建立索引会占用磁盘空间的索引文件

### 触发器

1、什么是触发器？

触发器是一种与表操作有关的数据库对象，当触发器所在表上出现指定事件时，将调用该对象，即表的操作事件触发表上的触发器的执行。

● CREATE TRIGGER <触发器名称> --触发器必须有名字，最多64个字符，可能后面会附有分隔符.它和MySQL中其他对象的命名方式基本相象.

● { BEFORE | AFTER } --触发器有执行的时间设置：可以设置为事件发生前或后。

● { INSERT | UPDATE | DELETE } --同样也能设定触发的事件：它们可以在执行insert、update或delete的过程中触发。

● ON <表名称> --触发器是属于某一个表的:当在这个表上执行插入、 更新或删除操作的时候就导致触发器的激活. 我们不能给同一张表的同一个事件安排两个触发器。

● FOR EACH ROW --触发器的执行间隔：FOR EACH ROW子句通知触发器 每隔一行执行一次动作，而不是对整个表执行一次。

2、MySQL包含对触发器的支持。

<触发器SQL语句> --触发器包含所要触发的SQL语句：这里的语句可以是任何合法的语句， 包括复合语句，但是这里的语句受的限制和函数的一样。

你必须拥有相当大的权限才能创建触发器（CREATE TRIGGER），如果你已经是Root用户，那么就足够了。这跟SQL的标准有所不同。

## Go程序操作MySQL数据库

### 安装mysql模块

1、go官方提供了database包

● database包下有sql/driver。该包用来定义操作数据库的接口，这保证了无论使用哪种数据库，他们的操作方式都是相同的。 ​

● 但go官方并没有提供连接数据库的driver，如果要操作数据库，还需要第三方的driver 包。

● 最常用的有： ​ https://github.com/Go-SQL-Driver/MySQL ，支持database/sql，全部采用go写。

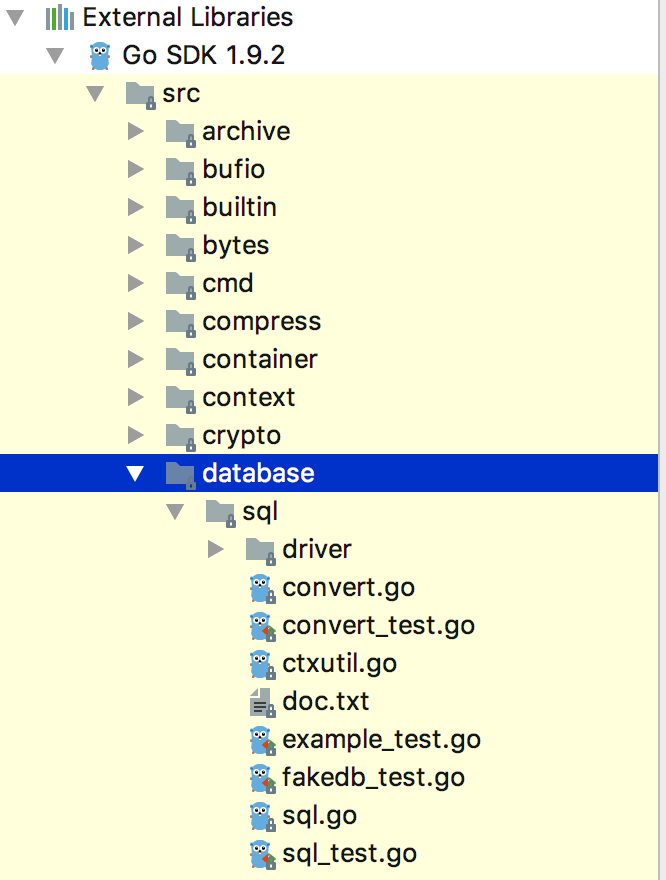


图 30

2、下载安装mysql驱动模块

（1）、Windows系统下载

● 打开cmd窗口输入：go get github.com/go-sql-driver/mysql，会下载到你的GOPATH路径的src 下

● 如果没有安装git，需要先安装git。否则安装失败。

（2）、Ubuntu下安装方式：

● 执行下面两个命令

● 下载：go get github.com/Go-SQL-Driver/MySQL

● 安装：go install github.com/Go-SQL-Driver/MySQL

● 如果没有安装git，需要先安装git。否则安装失败。

（3）、下载后目录结构

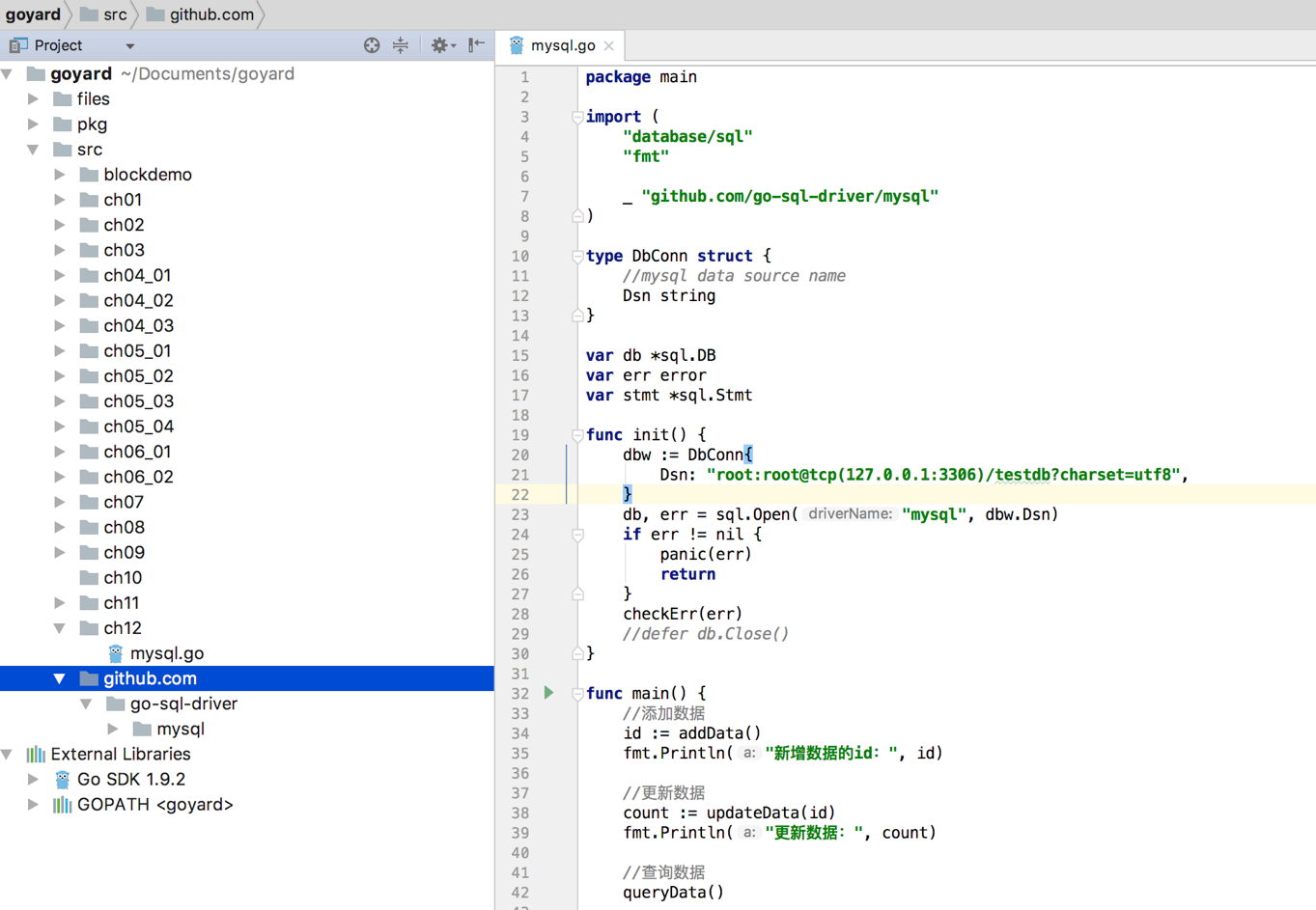


图 31

### 导入包

1、示例代码

import (

"database/sql"

"fmt"

\_ "github.com/go-sql-driver/mysql"

)

2、代码说明

● Golang 提供了database/sql包，用于对SQL数据库的访问。

● 它提供了一系列接口方法，用于访问关系数据库。它并不会提供数据库特有的方法，那些特有的方法交给数据库驱动去实现。

● 匿名导包——只导入包但是不使用包内的类型和数据

○ 使用匿名导包的方式(在包路径前添加下划线 “\_” )导入MySQL驱动。

○ 匿名导入的包与其他方式导入包一样，会让导入包编译到可执行文件中。

○ 通常来说, 导入包后就能调用该包中的数据和方法。但是对于数据库操作来说，我们不应该直接使用导入的驱动包所提供的方法，而是应该使用 sql.DB对象所提供的统一的方法。因此在导入 mysql 驱动时，使用了匿名导入包的方式。当导入一个数据库驱动后，该驱动会自行初始化并注册到Golang的database/sql上下文中，这样就可以通过 database/sql 包所提供的方法来访问数据库了。

### 连接数据库

1、sql包中的Open()函数

● sql.Open()函数原型如下:

func Open(driverName, dataSourceName string) (\*DB, error)

● driverName: 使用的驱动名. 这个名字其实就是数据库驱动注册到 database/sql 时所使用的名字.

● dataSourceName: 数据库连接信息，这个连接包含了数据库的用户名, 密码, 数据库主机以及需要连接的数据库名等信息.

● db, err := sql.Open("mysql", "用户名:密码@tcp(IP:端口)/数据库?charset=utf8")

● 例如：db, err := sql.Open("mysql", "root:123456@tcp(127.0.0.1:3306)/testdb?charset=utf8")

2、sql.Open()函数返回sql.DB对象

● 作为操作数据库的入口对象sql.DB, 主要为我们提供了两个重要的功能。

● sql.DB 通过数据库驱动为我们提供管理底层数据库连接的打开和关闭操作。

● sql.DB 为我们管理数据库连接池。正在使用的连接被标记为繁忙，用完后回到连接池等待下次使用。所以，如果你没有把连接释放回连接池，会导致过多连接使系统资源耗尽。

● sql.Open()返回的sql.DB对象是协程并发安全的。

● sql.DB的设计就是用来作为长连接使用的。不要频繁Open, Close。比较好的做法是，为每个不同的datastore建一个DB对象，保持这些对象Open。如果需要短连接，那么把DB作为参数传入function，而不要在function中Open, Close。

### 增删改数据

1、直接调用db对象的Exec()方法

● func (db \*DB) Exec(query string, args ...interface{}) (Result, error)

● 通过db.Exec()插入数据，通过返回的err可知插入失败的原因，通过返回的结果可以进一步查询本次插入数据影响的行数RowsAffected和最后插入的Id(如果数据库支持查询最后插入Id).

● 示例代码：

result, err := db.Exec("INSERT INTO userinfo (username, departname, created) VALUES (?, ?, ?)","Steven","区块链教学部","2017-10-1")

2、预编译语句(Prepared Statement)

● 预编译语句(PreparedStatement)提供了诸多好处, 因此我们在开发中尽量使用它.

● PreparedStatement 可以实现自定义参数的查询

● PreparedStatement 通常来说, 比手动拼接字符串 SQL 语句高效.

● PreparedStatement 可以防止SQL注入攻击

● 一般用Prepared Statements和Exec()完成INSERT, UPDATE, DELETE操作。

3、使用db对象的Prepare()方法获得预编译对象stmt，然后调用Exec()方法

● func (db \*DB) Prepare(query string) (\*Stmt, error)

● 示例代码：

○ stmt, err := db.Prepare("INSERT userinfo SET username=?,departname=?,created=?")

○ result, err := stmt.Exec("Jackson", "研发部", "2017-10-1")

● 获取影响数据库的行数，可以根据该数值判断是否操作（插入、删除或修改）成功。

○ count, err := result.RowsAffected()

● 获得刚刚添加数据的自增ID

○ id, err := result.LastInsertId()

### 查询数据

1、数据库查询的一般步骤如下

● 调用 db.Query()方法执行 SQL 语句, 此方法返回一个 Rows 作为查询结果；

○ func (db \*DB) Query(query string, args ...interface{}) (\*Rows, error)

● 将rows.Next()方法的返回值作为for循环的条件，迭代查询数据；

○ func (rs \*Rows) Next() bool

● 在循环中，通过 rows.Scan()方法读取每一行数据；

○ func (rs \*Rows) Scan(dest ...interface{}) error

● 调用db.Close()关闭查询。

2、通过QueryRow()方法查询单条数据

● func (db \*DB) QueryRow(query string, args ...interface{}) \*Row

● 示例代码：

○ var username, departname, created string

err := db.QueryRow("SELECT username,departname,created FROM user\_info WHERE uid=?", 3).Scan(&username, &departname, &created)

3、查询多行数据

● stmt, err := db.Prepare("SELECT \* FROM user\_info WHERE uid<?")

● rows, err := stmt.Query(10)

● user := new(UserTable)

for rows.Next() {

err := rows.Scan(&user.Uid, &user.Username, &user.Department, &user.Created)

if err != nil {

panic(err)

continue

}

fmt.Println(\*user)

}

4、查询数据时需要注意的细节

● rows.Scan()方法的参数顺序很重要，必须和查询结果的column相对应（数量和顺序都需要一致）。

○ 例如 “SELECT \* From user\_info where age >=20 AND age < 30” 查询的column 顺序是 “id, name, age” 和插入操作顺序相同, 因此 rows.Scan() 也需要按照此顺序 rows.Scan(&id, &name, &age), 不然会造成数据读取的错位。

● 因为golang是强类型语言，所以查询数据时先定义数据类型。

○ 查询数据库中的数据存在三种可能：存在值、存在零值、未赋值NULL 三种状态，因此可以将待查询的数据类型定义为sql.NullString 、 sql.NullInt64类型等。

○ 可以通过判断Valid值来判断查询到的值是否为赋值状态还是未赋值NULL状态。

● 每次db.Query操作后, 都建议调用rows.Close()。

○ 因为 db.Query() 会从数据库连接池中获取一个连接, 这个底层连接在结果集(rows)未关闭前会被标记为处于繁忙状态。当遍历读到最后一条记录时，会发生一个内部EOF错误，自动调用rows.Close()。但如果出现异常，提前退出循环，rows不会关闭，连接不会回到连接池中，连接也不会关闭, 则此连接会一直被占用。

○ 因此通常使用 defer rows.Close() 来确保数据库连接可以正确放回到连接池中。

○ 阅读源码发现rows.Close()操作是幂等操作，而一个幂等操作的特点是：其任意多次执行所产生的影响与一次执行的影响相同。所以即便对已关闭的rows再执行close()也没关系。

### 示例代码

1、案例中的表结构

CREATE TABLE user\_info (

uid int(10) NOT NULL AUTO\_INCREMENT,

username varchar(64) DEFAULT NULL,

departname varchar(64) DEFAULT NULL,

created date DEFAULT NULL,

PRIMARY KEY (uid)

) ;

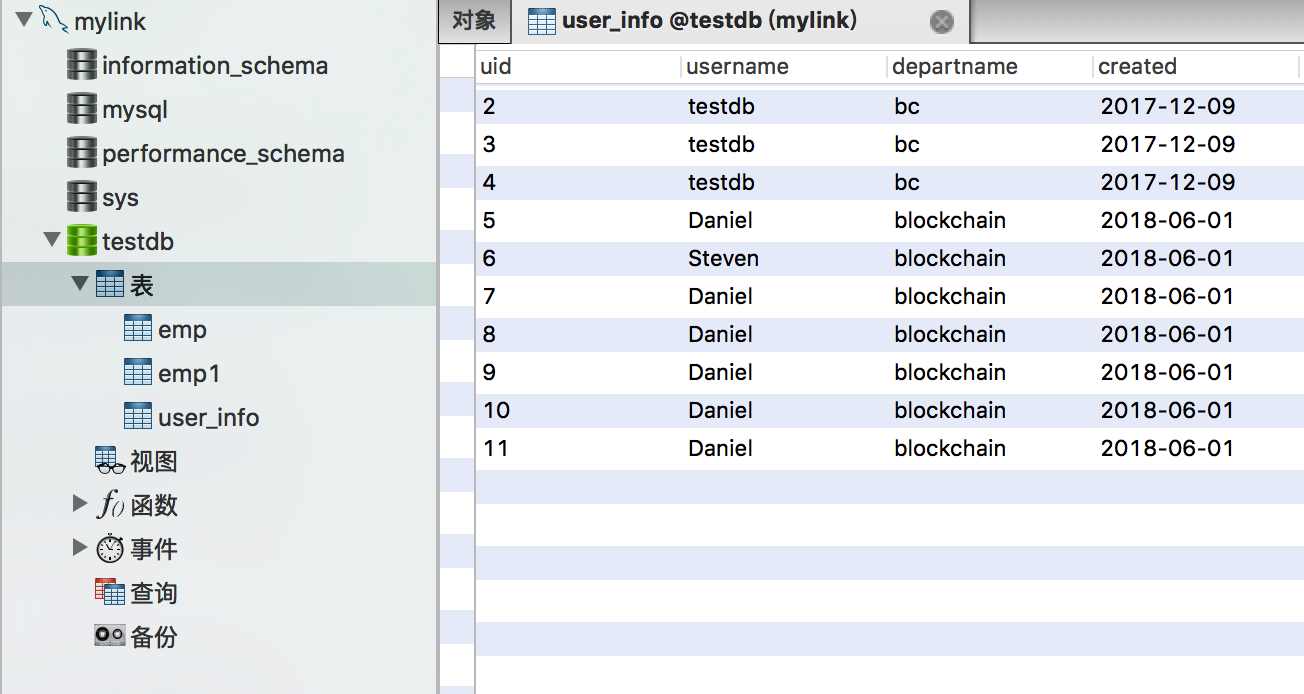


图 32

2、完整示例代码

package main

import (

"database/sql"

"fmt"

\_ "github.com/go-sql-driver/mysql"

)

//定义数据库连接信息

type DbConn struct {

Dsn string //数据库驱动连接字符串

Db \*sql.DB

}

//user\_info表的映射对象

type UserTable struct {

Uid int

Username string

Department string

Created string

}

func main() {

var err error

dbConn := DbConn{

Dsn: "root:root@tcp(127.0.0.1:3306)/testdb?charset=utf8",

}

dbConn.Db, err = sql.Open("mysql", dbConn.Dsn)

if err != nil {

panic(err)

return

}

defer dbConn.Db.Close()

//1、测试封装的ExecData()方法

//dbConn.execData(&dbConn)

//2、测试封装的PreExecData()方法

//dbConn.preExecData(&dbConn)

//3、查询单行数据

//查询最后一条数据的信息

//result := dbConn.QueryRowData("select \* from user\_info where uid=(select max(uid) from user\_info)")

//fmt.Println(result)

//4、查询多行数据

//result := dbConn.QueryData("select \* from user\_info where uid<10")

//fmt.Println(len(result))

////遍历查询的结果集

//for k, v := range result {

// fmt.Println("uid：", k, v)

//}

//5、查询多行数据

result := dbConn.PreQueryData("select \* from user\_info where uid<? order by uid desc" , 10)

fmt.Println(len(result))

//遍历查询的结果集

for k, v := range result {

fmt.Println("uid：", k, v)

}

}

//一、测试封装的ExecData()函数

func execData(dbConn \*DbConn) {

count, id, err := dbConn.ExecData("INSERT user\_info(username , departname , created) VALUES ('Josh','business group','2018-07-3')")

//count , err := execData("UPDATE user\_info SET created='2018-06-30' WHERE uid=14")

//count , err := execData("DELETE FROM user\_info WHERE uid=10")

if err != nil {

fmt.Println(err.Error())

} else {

fmt.Println("受影响行数：", count)

fmt.Println("新添加数据的id：", id)

}

}

//二、测试封装的PreExecData()函数

func preExecData(dbConn \*DbConn) {

count, id, err := dbConn.PreExecData("INSERT user\_info(username , departname , created) VALUES (?,?,?)", "Jackson", "Education Department", "2017-10-8")

//count, id, err := PreExecData("Delete from user\_info WHERE uid<?", 4)

//count, id, err := PreExecData("UPDATE user\_info set departname=? WHERE departname = ?", "BC Group", "blockchain")

if err != nil {

fmt.Println(err.Error())

} else {

fmt.Println("受影响行数：", count)

fmt.Println("新添加数据的id：", id)

}

}

//1、封装增删改数据的函数，该函数直接使用DB的Exec()方法实现数据操作

func (dbConn \*DbConn) ExecData(sqlString string) (count, id int64, err error) {

result, err := dbConn.Db.Exec(sqlString)

if err != nil {

panic(err)

return

}

if id, err = result.LastInsertId(); err != nil {

panic(err)

return

}

if count, err = result.RowsAffected(); err != nil {

panic(err)

return

}

return count, id, nil

}

//2、封装增删改数据的函数，该函数使用预编译语句加Exec()方法实现增删改数据

func (dbConn \*DbConn) PreExecData(sqlString string, args ...interface{}) (count, id int64, err error) {

stmt, err := dbConn.Db.Prepare(sqlString)

defer stmt.Close()

if err != nil {

panic(err)

return

}

result, err := stmt.Exec(args ...)

if err != nil {

panic(err)

return

}

if id, err = result.LastInsertId(); err != nil {

panic(err)

return

}

if count, err = result.RowsAffected(); err != nil {

panic(err)

return

}

return count, id, nil

}

//3、查询当行数据

func (dbConn \*DbConn) QueryRowData(sqlString string) (data UserTable) {

user := new(UserTable)

err := dbConn.Db.QueryRow(sqlString).Scan(&user.Uid, &user.Username, &user.Department, &user.Created)

if err != nil {

panic(err)

return

}

return \*user

}

//4、未使用预编译，直接查询多行数据

func (dbConn \*DbConn) QueryData(sqlString string) (resultSet map[int]UserTable) {

rows, err := dbConn.Db.Query(sqlString)

defer rows.Close()

if err != nil {

panic(err)

return

}

resultSet = make(map[int]UserTable)

user := new(UserTable)

for rows.Next() {

err := rows.Scan(&user.Uid, &user.Username, &user.Department, &user.Created)

if err != nil {

panic(err)

continue

}

resultSet[user.Uid] = \*user

}

return resultSet

}

//5、使用预编译语句进行查询多行数据

func (dbConn \*DbConn) PreQueryData(sqlString string , args ...interface{}) (resultSet map[int]UserTable) {

stmt, err := dbConn.Db.Prepare(sqlString)

defer stmt.Close()

if err != nil {

panic(err)

return

}

rows, err := stmt.Query(args ...)

defer rows.Close()

if err != nil {

panic(err)

return

}

resultSet = make(map[int]UserTable)

user := new(UserTable)

for rows.Next() {

err := rows.Scan(&user.Uid, &user.Username, &user.Department, &user.Created)

if err != nil {

panic(err)

continue

}

resultSet[user.Uid] = \*user

}

return resultSet

}

//无返回值，只打印输出，用于测试

func (dbConn \*DbConn) PreQueryData2(sqlString string , args ...interface{}) {

stmt, err := dbConn.Db.Prepare(sqlString)

defer stmt.Close()

if err != nil {

panic(err)

return

}

rows, err := stmt.Query(args ...)

defer rows.Close()

if err != nil {

panic(err)

return

}

user := new(UserTable)

for rows.Next() {

err := rows.Scan(&user.Uid, &user.Username, &user.Department, &user.Created)

if err != nil {

panic(err)

continue

}

fmt.Println(\*user)

}

}