巍巍交大　百年书香
www.jiaodapress.com.cn
bookinfo@sjtu.edu.cn

丛书策划　张荣昌
责任编辑　王　清　孟海江
封面设计

大数据、云计算、人工智能、信息安全人才培养丛书
“互联网+”新形态一体化教材

大数据
- 大数据基础
- 数据可视化
- 数据清洗与治理
- Hadoop应用与开发（第2版）
- 数据挖掘基础
- SEO搜索引擎优化
- R语言程序设计
- Go语言程序设计

云计算
- 云计算基础
- 虚拟化与容器
- 云安全运维
- 网络工程与组网技术
- 现代通信技术
- 路由交换技术
- 无线网络技术
- 现代网络SDN技术
- 数据网组建与维护
- 局域网组建与维护
- 云数据中心架构与SDN技术

人工智能
- 人工智能基础
- 物联网基础
- 深度学习
- 机器学习

信息安全
- 信息安全基础
- Linux服务器安全高级运维
- Web安全与防御
- 防火墙技术与应用
- 计算机病毒与防范
- 数据存储与恢复
- 密码学基础
- 计算机网络安全运维
- 网络设备配置与综合实战
- 无线网络安全技术
- VPN虚拟专用网安全
- 终端数据存储与恢复
- 工控安全
- 渗透测试
- 恶意代码分析
- 网络空间安全态势感知
- 数据库安全技术
- 网络安全协议
- 企业级数据安全与灾备管理
- 信息安全法律法规
- 终端数据安全及防泄密

专业基础
- Linux操作系统基础
- 计算机网络基础
- Ubuntu服务器管理
- Windows Server 配置与管理
- 数据结构
- Python程序设计
- Java程序设计
- C语言程序设计
- C#程序设计
- Android程序设计
- XML基础教程
- JavaScript基础教程
- Web前端开发
- OpenStack应用与开发
- Spark应用与开发
- 静态网页设计与制作
- HTML5与JavaScript程序设计
- MySQL数据库原理与应用（第2版）
- 数据库应用基础
- UML建模与设计模式
- ERP原理与应用
- 综合布线

本书提供教学资源包
网址：https://www.sjhtbook.com

扫描二维码
关注上海交通大学出版社
官方微信

ISBN 978-7-313-31848-0
9 787313 318480 >
定价：58.00元

广东省『十四五』职业教育规划教材　MySQL数据库原理与应用（第2版）　主编◎李　蓉　管芳景　梁　宾

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

广东省“十四五”职业教育规划教材

MySQL数据库原理与应用

（第2版）

主编◎李　蓉　管芳景　梁　宾

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

广东省“十四五”职业教育规划教材

MySQL数据库原理与应用

（第2版）

主　编◎李　蓉　管芳景　梁　宾
副主编◎卜　旭　黄　锵　廖福保
李冬睿　邓会敏　邱尚明

扫一扫
学习资源库

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

本书遵循学习成果导向理念，按照数据库设计、开发、应用的顺序，引领学习者熟悉数据库的体系结构，领会数据库的基本原理，掌握数据库的实现技术。本书充分体现“项目引领、实践导向”的思想，以一个实际数据库系统项目为主线，内容包括数据库开发环境的安装与配置、数据库的设计与创建、数据库对象的建立与管理、数据操作与查询、数据库的管理与维护等，以实现中小型数据库应用系统的设计与开发，帮助学习者更好地对接企业岗位的真实需求。本书可作为高等院校计算机类相关专业的教学用书，也可作为相关技术人员培训或工作的参考用书。

图书在版编目（CIP）数据

MySQL 数据库原理与应用 / 李蓉，管芳景，梁宾主编．2 版．-- 上海 ：上海交通大学出版社，2025.1 -- ISBN 978-7-313-31848-0

Ⅰ．TP311.132.3

中国国家版本馆 CIP 数据核字第 2024R7H378 号

MySQL 数据库原理与应用（第 2 版）

MySQL SHUJUKU YUANLI YU YINGYONG（DI 2 BAN）

主　　编：李蓉　管芳景　梁宾
出版发行：上海交通大学出版社
邮政编码：200030
印　　制：北京荣玉印刷有限公司
开　　本：889 mm × 1194 mm　1/16
字　　数：424 千字
版　　次：2022 年 2 月第 1 版
　　2025 年 1 月第 2 版
书　　号：ISBN 978-7-313-31848-0
定　　价：58.00 元

地　　址：上海市番禺路 951 号
电　　话：021-6407 1208
经　　销：全国新华书店
印　　张：17
印　　次：2025 年 1 月第 5 次印刷
电子书号：ISBN 978-7-89424-939-5

前言

党的二十大报告明确指出，要加快建设数字中国，加快发展数字经济，促进数字经济和实体经济深度融合。随着数字化转型的深入推进和数据量的爆炸式增长，各行各业的应用对数据库的需求变化推动数据库技术加速创新，全球数据库产业快速发展，我国已迈入第一梯队。根据中国通信标准化协会大数据技术标准推进委员会发布的《数据库发展研究报告（2024 年）》，预计到 2028 年，中国数据库市场总规模将达到 930.29 亿元，数据库在我国的信息化、数字化发展和建设中正发挥着越来越重要的作用。

数据库是信息系统的核心软件，向下调动系统资源，向上支撑应用软件，承担着管理、组织、存储和计算数据的重任。从产业链视角来看，数据库处在中游位置，是 IT 产业的重要枢纽，地位举足轻重。随着大数据、人工智能、区块链、5G 等数字技术的兴起，各行各业数字化转型不断加速，我国数据库应用创新实践迈入新阶段，数据库技术人才面临着空前的需求和挑战，数据库技术课程已成为高等院校计算机等相关专业必修的核心课程。

本书遵循学习成果导向理念，结合企业实际工作过程，采用“项目引领、任务驱动、实践导向”的模式，以“教务管理系统”数据库为主线，包括认识数据库、安装与配置 MySQL、创建和管理数据库、创建和管理数据表、数据操作、单表数据查询、多表数据查询、索引、视图、数据库编程、安全性管理、事务管理、数据库备份与还原、图书管理系统的设计与开发，突出体系新颖、技术先进、通俗易懂、实用性强等特点，帮助学习者熟悉数据库的体系结构，领会数据库的基本原理，掌握数据库的实现技术，更好地对接企业岗位的真实需求。

本书秉持德技并修的育人理念，设计专业育才与思政育人两条主线，全面贯彻党的二十大精神，落实立德树人根本任务。在专业学习方面，通过分析企业技术需求所涵盖的岗位任务和职业能力，系统梳理知识点之间的逻辑关系和先后次序，开发 14 个学习项目，通过“学习成果目标”“学习任务”“学习内容”“任务实践”“拓展阅读”“思考与练习”几个部分，循序渐进地开展理实一体化的项目式教学，做到分阶段训练、分层次深入、分维度培养。在思政教育方面，注重对学习者家国情怀、职业精神、规范意识、工程意识等思想性、价值性品质的培养，引导学习者树立正确的荣誉观、职业观、工程观和方法观。

本书可以作为计算机等相关专业数据库课程的教材，也可供从事数据库系统研究、开发和应用的研究人员或工程技术人员参考，还可以作为全国计算机等级考试二级 MySQL 数据库程序设计、全国计算机技术与软件专业技术资格（水平）考试“数据库系统工程师”、“1+X”Web 前端开发职业技能等级证书等的考试参考书。我们希望读者不仅阅读本书的内容，还动手实操书中的案例，充分锻炼自己实际动手的能力，深化对理论知识的思考和理解，达到理论联系实际的学习效果。

此外，编者还为广大一线教师提供了服务于本书的教学资源库，有需要者可发邮件至

2393867076@qq.com。

在编写本书的过程中，我们得到了领导和同行的大力支持和帮助，在此表示诚挚的感谢。同时，非常感谢数据库课程团队的同仁，他们对本书的修订提出了积极的意见和建议，使本书得以顺利完成。

本书是老师们多年数据库教学和实践工作经验的总结，在编写过程中我们尽可能引入新的技术和方法，力求反映当前的技术水平和未来的发展方向。尽管我们付出了很大的努力，但书中仍有欠妥之处，真诚欢迎读者和同行们对本书提出宝贵的意见和建议，我们将不胜感激！

目录

项目 4 创建和管理数据表

学习成果目标

知识目标

（1）理解数据表的基本知识。

（2）熟悉 MySQL 数据类型。

（3）理解数据完整性的概念。

（4）理解完整性约束条件的作用。

（5）掌握创建、查看、修改、删除数据表的方法。

能力目标

（1）能够根据实际数据需求设计和调整数据表结构。

（2）能够根据用户需求为数据表设置合理的约束条件。

（3）能够熟练使用 SQL 语句创建和管理数据表。

素质目标

（1）培养勇于实践、大胆求知的品质。

（2）培养不忘初衷、坚持不懈的精神。

（3）锻炼数学思维和计算思维。

学习任务

在创建好的教务管理系统数据库“EduSys”中，创建“Student”“Course”“Teacher”“Class”“StuCourse”和“TeaCourse”六个数据表，分别存放学生信息、课程信息、教师信息、班级信息、学生选课信息和教师授课信息。各数据表的结构如表 4-1 至表 4-6 所示。

表 4-1　Student（学生信息表）

字段名称	类型	宽度	允许空值	要求	说明
Sno	char	8	NOT NULL	主键	学生学号
Sname	varchar	20	NOT NULL	—	姓名
Sex	char	2	NULL	取值范围“男”和“女”	性别
Native	varchar	30	NULL	—	籍贯
Birthday	datetime	—	NULL	—	出生日期
Major	varchar	20	NULL	—	所在专业
Classno	char	4	NULL	外键	班级编号
Tel	char	11	NULL	—	联系电话

笔记

表 4-2　Course（课程信息表）

字段名称	类型	宽度	允许空值	要求	说明
Cno	char	5	NOT NULL	主键	课程编号
Cname	varchar	30	NOT NULL	—	课程名称
Hours	tinyint	—	NULL	—	课程学时
Credit	tinyint	—	NULL	—	课程学分
Semester	tinyint	—	NULL	—	开课学期

表 4-3　Teacher（教师信息表）

字段名称	类型	宽度	允许空值	要求	说明
Tno	char	8	NOT NULL	主键	教师编号
Tname	varchar	20	NOT NULL	—	教师姓名
Sex	char	2	NULL	取值范围“男”和“女”	教师性别
Birthday	datetime	—	NULL	—	教师出生日期
Dno	varchar	20	NULL	—	教师所在院系
Pno	varchar	10	NULL	—	教师职称
Tel	char	11	NULL	唯一	联系电话
Email	varchar	40	NULL	唯一	电子邮件

表 4-4　Class（班级表）

字段名称	类型	宽度	允许空值	要求	说明
Classno	char	4	NOT NULL	主键	班级编号
Classname	char	16	NOT NULL	—	班级名称
Num	int	—	NULL	—	人数
Charge	varchar	20	NULL	—	班主任

表 4-5　StuCourse（学生选课表）

字段名称	类型	宽度	允许空值	要求	说明
Sno	char	8	NOT NULL	主键、外键	学生学号
Cno	char	5	NOT NULL	主键、外键	课程编号
Score	tinyint	—	NULL	—	学生成绩

表 4-6　TeaCourse（教师授课表）

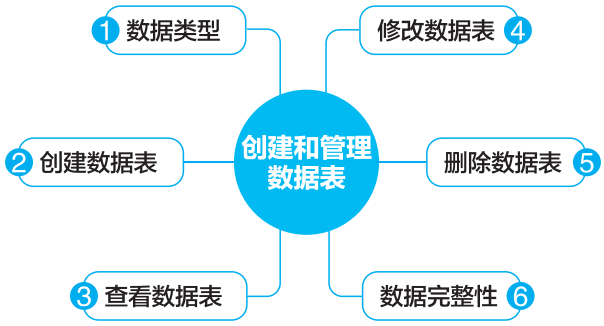
字段名称	类型	宽度	允许空值	要求	说明
Tno	char	8	NOT NULL	主键、外键	教师编号
Classno	char	4	NOT NULL	主键、外键	班级编号
Cno	char	5	NOT NULL	主键、外键	课程编号
Semester	char	6	NULL	—	学期
Schoolyear	char	10	NULL	—	学年

学习内容

笔记

数据库本身无法直接存储数据，存储数据是通过数据库中的数据表来实现的。数据表是数据库存放数据的对象实体，没有数据表，数据库中其他的数据对象就没有意义。因此，在创建数据库后，就应该创建数据表。通常数据库中包含多张表，以存储用户需求的数据。对数据表的操作主要包括数据表的创建、查看、修改和删除等。

本项目的内容结构如下：



4.1 数据类型

数据类型（data type）是指数据库中表列、存储过程参数、表达式和局部变量的数据特征，它决定了数据的存储格式，代表了不同的信息类型。MySQL 数据类型定义了列中可以存储什么数据，以及该数据怎样存储的规则。

数据库中的每个列都应该有适当的数据类型，用于限制或允许该列中存储的数据。错误的数据类型可能会严重影响应用程序的功能和性能，例如，列中存储的数据是数字，则相应的数据类型应该为数值类型。修改已经包含数据的数据类型须特别小心，否则可能会导致数据丢失或系统错误。所以在设计表时，应该特别重视数据列所用的数据类型，在创建表时就为每个列设置合适的数据类型和长度。

MySQL 的数据类型分为多种，包括整数类型、浮点数类型、定点数类型、时间日期类型、字符串类型、二进制类型等。整数类型、浮点数类型、定点数类型可以统称为数值类型。接下来对 MySQL 的数据类型做详细的介绍。

4.1.1 整数类型

整数类型又称数值型数据类型，数值型数据类型主要用来存储数字。MySQL 提供了多种数值型数据类型，不同的类型提供不同的取值范围，可以存储的值范围越大，所需的存储空间也会越大。整数类型主要有 TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT，如表 4-7 所示。设置为整数类型的字段可以添加 AUTO_INCREMENT 自增约束条件。

笔记

表 4-7　MySQL 中的数值类型

类型名称	说明	存储需求
TINYINT	很小的整数	1 个字节
SMALLINT	小的整数	2 个字节
MEDIUMINT	中等大小的整数	3 个字节
INT（INTEGHR）	普通大小的整数	4 个字节
BIGINT	大整数	8 个字节

从表 4-7 中可以看到，不同类型的整数存储所需的字节数不相同，占用字节数最小的是 TINYINT 类型，占用字节数最大的是 BIGINT 类型，占用字节越多的类型所能表示的数值范围越大。根据占用字节数可以求出每一种数据类型的取值范围。例如，TINYINT 需要 1 个字节（8bit）来存储，那么 TINYINT 无符号数的最大值为 2^8-1，即 255；TINYINT 有符号数的最大值为 2^7-1，即 127。不同整数类型的取值范围如表 4-8 所示。

表 4-8　MySQL 中数值类型的取值范围

类型名称	取值范围（有符号数）	取值范围（无符号数）
TINYINT	–128 ~ 127	0 ~ 255
SMALLINT	–32 768 ~ 32 767	0 ~ 65 535
MEDIUMINT	–8 388 608 ~ 8 388 607	0 ~ 16 777 215
INT（INTEGER）	–2 147 483 648 ~ 2 147 483 647	0 ~ 4 294 967 295
BIGINT	–9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807	0 ~ 18 446 744 073 709 551 615

注意：

显示宽度与数据类型的取值范围是无关的。显示宽度是指 MySQL 最大可能显示的数字个数，数值的位数小于指定的宽度时会由空格填充。如果插入了大于显示宽度的值，只要该值不超过该类型整数的取值范围，数值依然可以插入，而且能够显示出来。例如，Year 字段插入 19999，当使用 SELECT 查询该列值的时候，MySQL 显示的将是完整的带有 5 位数字的 19999，而不是 4 位数字的值。其他整型数据类型也可以在定义表结构时指定所需的显示宽度，如果不指定，则系统为每一种类型指定默认的宽度值。

4.1.2 小数类型

MySQL 中使用浮点数和定点数来表示小数。浮点类型有两种，分别是单精度浮点数（FLOAT）和双精度浮点数（DOUBLE）；定点数类型有一种，是 DECIMAL。浮点数类型和定点数类型都可以用（*M*，*D*）来表示，其中 *M* 称为精度，表示总共的位数；*D* 称为标度，表示小数位数。浮点数类型的取值范围为 *M*（1 ~ 255）和 *D*（1 ~ 30，且不能大于 *M*–2），*M* 和 *D* 在 FLOAT 和 DOUBLE 中是可选的，默认 FLOAT 和 DOUBLE 类型将

被保存为硬件所支持的最大精度。定点数类型的取值范围为 M（1 ~ 65）和 D（1 ~ 30），DECIMAL 的默认 M 值为 10、D 值为 0。表 4-9 中列出了 MySQL 中的小数类型和存储需求。

笔记

表 4-9　MySQL 中的小数类型和存储需求

类型名称	说明	存储需求
FLOAT	单精度浮点数	4 个字节
DOUBLE	双精度浮点数	8 个字节
DECIMAL（M，D）	压缩的“严格”定点数	M+2 个字节

从表 4-9 中可知，FLOAT 有符号类型取值范围为 –3.402 823 466E+38 ~ –1.175494351E–38、0 及 1.175 494 351E–38 ~ 3.402 823 466 351E+38；无符号类型取值范围为 0 和 1.175 494 351E–38 ~ 3.402 823 466E+38，FLOAT 类型只保证 6 位有效数字的准确性。DOUBLE 有符号类型取值范围为 –1.797 693 134 862 315 7E+308 ~ –2.225 073 858 507 201 4E–308、0 及 2.225 073 858 507 201 4E–308 ~ 1.797 693 134 862 315 7E+308；无符号类型取值范围为 0 和 2.225 073 858 507 201 4E–308 ~ 1.797 693 134 862 315 7E+308，DOUBLE 类型只保证 15 位有效数字的准确性。DECIMAL 的存储空间不固定，DECIMAL 可能的最大取值范围与 DOUBLE 相同，但是有效的取值范围由 M 和 D 决定，如果改变 M 而固定 D，则取值范围将随 M 的变大而变大，在对精度要求比较高的时候（如货币、科学数据），使用 DECIMAL 类型比较好。

不论是定点数类型还是浮点数类型，如果用户指定的精度超出精度范围，MySQL 会进行四舍五入。FLOAT 和 DOUBLE 不指定精度时，默认为实际的精度（由计算机硬件和操作系统决定）；DECIMAL 不指定精度时，默认为（10，0）。浮点数相对于定点数的优点是，在长度一定的情况下，浮点数能够表示更大的范围；缺点是浮点数是不准确的，会引起精度问题，所以要避免使用“=”判断两个数是否相等。

4.1.3 时间日期类型

MySQL 有多种表示日期的数据类型，包括 YEAR、TIME、DATE、DTAETIME、TIMESTAMP 等。每一个类型都有合法的取值范围，表 4-10 列出了 MySQL 中的时间日期类型的具体情况。

表 4-10　MySQL 中的时间日期类型

类型名称	日期格式	日期范围	存储需求
YEAR	YYYY	1901 ~ 2155	1 个字节
TIME	HH:MM:SS	-838:59:59 ~ 838:59:59	3 个字节
DATE	YYYY-MM-DD	1000-01-01 ~ 9999-12-3	3 个字节
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	8 个字节
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1980-01-01 00:00:01 UTC ~ 2040-01-19 03:14:07 UTC	4 个字节

笔记

1.YEAR 类型

YEAR 类型是一个单字节类型，用于表示年，在存储时只需要 1 个字节。使用时可以用各种格式指定 YEAR，以 4 位字符串或者 4 位数字格式表示的 YEAR，范围为 '1901' ~ '2155'，输入格式为 'YYYY' 或者 YYYY。例如，输入 '2010' 或 2010，插入数据库的值均为 2010。以 2 位字符串格式表示的 YEAR，范围为 '00' 到 '99'，'00' ~ '69' 和 '70' ~ '99' 的值分别被转换为 2000 ~ 2069 和 1970 ~ 1999 范围的 YEAR 值。'0' 与 '00' 的作用相同，插入该值将被转换为 2000。以 2 位数字表示的 YEAR，范围为 1 ~ 99，1 ~ 69 和 70 ~ 99 范围的值分别被转换为 2001 ~ 2069 和 1970 ~ 1999 范围的 YEAR 值。

注意：

两位整数范围与两位字符串范围稍有不同。例如，要插入 2000 年，读者可能会使用数字格式的 0 表示 YEAR，实际上，此时插入数据库的值为 0000，而不是 2000。只有使用字符串格式的 '0' 或 '00'，才可以被正确解释为 2000。非法 YEAR 值将被转换为 0000。

2.TIME 类型

TIME 类型用于只需要时间信息的值，在存储时需要 3 个字节。格式为 HH:MM:SS，HH 表示小时，MM 表示分钟，SS 表示秒。TIME 类型的取值范围为 –838:59:59 ~ 838:59:59，小时部分如此大的原因是 TIME 类型不仅可以用于表示一天的时间（必须小于 24 小时），还可能是某个事件过去的时间或两个事件之间的时间间隔（可大于 24 小时，或者为负）。

可以使用各种格式指定 TIME 值，如 'D HH:MM:SS' 格式的字符串或 'HH:MM:SS'、'HH:MM'、'D HH'、'SS' 等“非严格”语法，这里的 D 表示日，可取 0 ~ 34 之间的值。在插入数据库时，D 会被转换为小时保存，格式为“D*24+HH”。'HHMMSS' 格式、没有间隔符的字符串或者 HHMMSS 格式的数值，系统在进行理解时会假定为有意义的时间，例如，'101112' 被理解为 '10:11:12'，但是 '106112' 是不合法的（它有一个没有意义的分钟部分），在存储时将变为 00:00:00。

提示：

为 TIME 列分配简写值时应注意，如果没有冒号，MySQL 在解释值时会假定最右边的两位表示秒。例如，读者可能认为 '1112' 和 1112 表示 11:12:00（即 11 点过 12 分钟），但 MySQL 会将它们解释为 00:11:12（即 11 分 12 秒）。同样，'12' 和 12 被解释为 00:00:12。相反，TIME 值中如果使用冒号则被看作当天的时间，也就是说，'11:12' 表示 11:12:00，而不是 00:11:12。

3.DATE 类型

DATE 类型用于仅需要日期值时，没有时间部分，在存储时需要 3 个字节。格式为 'YYYY-MM-DD'，其中 YYYY 表示年，MM 表示月，DD 表示日。

在给 DATE 类型的字段赋值时，可以使用字符串类型或者数字类型的数据插入，只要

符合 DATE 的日期格式即可。以 'YYYY-MM-DD' 或者 'YYYYMMDD' 字符串格式表示的日期，取值范围为 1000-01-01 ~ 9999-12-3，例如，输入 '2015-12-31' 或者 '20151231'，插入数据库的日期为 2015-12-31。以 'YY-MM-DD' 字符串格式表示日期，在这里 YY 表示两位的年值，MySQL 按如下方法解释两位年值的规则：00 ~ 69 范围的年值转换为 2000 ~ 2069，70 ~ 99 范围的年值转换为 1970 ~ 1999。例如，输入 '15-12-31'，插入数据库的日期为 2015-12-31 ；输入 '991231'，插入数据库的日期为 1999-12-31。以 YYMMDD 数字格式表示的日期，与前面相似，00 ~ 69 范围的年值转换为 2000 ~ 2069，70 ~ 99 范围的年值转换为 1970 ~ 1999。例如，输入 151231，插入数据库的日期为 2015-12-31，输入 991231，插入数据库的日期为 1999-12-31。

使用 CURRENT_DATE 或者 NOW( )，可以插入当前系统日期。

提示：

MySQL 允许“不严格”语法，任何标点符号都可以用作日期部分之间的间隔符。例如，'98@11@31'、'98.11.31'、'98/11/31' 和 '98-11-31' 是等价的，这些值都可以正确地插入数据库。

4.DATETIME 类型

DATETIME 类型用于需要同时包含日期和时间信息的值，在存储时需要 8 个字节。格式为 'YYYY-MM-DD HH:MM:SS'，其中 YYYY 表示年，MM 表示月，DD 表示日，HH 表示小时，MM 表示分钟，SS 表示秒。

在给 DATETIME 类型的字段赋值时，可以使用字符串类型或者数字类型的数据插入，只要符合 DATETIME 的日期格式即可。以 'YYYY-MM-DD HH:MM:SS' 或者 'YYYYMMDDHHMMSS' 字符串格式表示的日期，取值范围为 1000-01-01 00:00:00 ~ 9999-12-3 23:59:59。例如，输入 '2014-12-31 05:05:05' 或者 '20141231050505'，插入数据库的 DATETIME 值都为 2014-12-31 05:05:05。

以 'YY-MM-DD HH:MM:SS' 或者 'YYMMDDHHMMSS' 字符串格式表示的日期，在这里 YY 表示两位的年值。与前面相同，00 ~ 69 范围的年值转换为 2000 ~ 2069，70 ~ 99 范围的年值转换为 1970 ~ 1999。例如，输入 '14-12-31 05:05:05'，插入数据库的 DATETIME 值为 2014-12-31 05:05:05 ；输入 141231050505，插入数据库的 DATETIME 值为 2014-12-31 05:05:05。

YYYYMMDDHHMMSS 或者 YYMMDDHHMMSS 数字格式也可以表示日期和时间。例如，输入 20141231050505，插入数据库的 DATETIME 值为 2014-12-31 05:05:05 ；输入 141231050505，插入数据库的 DATETIME 值为 2014-12-31 05:05:05。

5.TIMESTAMP 类型

TIMESTAMP 的显示格式与 DATETIME 相同，显示宽度固定在 19 个字符，日期格式为 YYYY-MM-DD HH:MM:SS，在存储时需要 4 个字节。但是 TIMESTAMP 列的取值范围小于 DATETIME 的取值范围，为 1970-01-01 00:00:01UTC ~ 2038-01-19 03:14:07UTC。在插入数据时，要保证在合法的取值范围内。

笔记

提示：

协调世界时（英：coordinated universal time，法：temps universal coordonné）又称为世界统一时间、世界标准时间、国际协调时间。英文（CUT）和法文（TUC）的缩写不同，作为妥协，简称 UTC。

TIMESTAMP 与 DATETIME 除了存储字节和支持的范围不同外，还有一个最大的区别：DATETIME 在存储日期数据时，按实际输入的格式存储，即输入什么就存储什么，与时区无关；而 TIMESTAMP 值的存储是以 UTC（协调时间时）格式保存的，存储时对当前时区进行转换，检索时再转换回当前时区。即查询时，根据当前时区的不同，显示的时间值是不同的。

提示：

如果为一个 DATETIME 或 TIMESTAMP 对象分配一个 DATE 值，结果值的时间部分被设置为 '00：00：00'，则 DATE 值未包含时间信息。如果为一个 DATE 对象分配一个 DATETIME 或 TIMESTAMP 值，结果值的时间部分被删除，则 DATE 值未包含时间信息。

4.1.4 字符串类型

字符串类型除了用来存储字符串数据，还可以存储图片、声音等二进制数据。字符串类型数据可以进行区分或者不区分大小写的字符串比较，还可以进行正则表达式的匹配查找。MySQL 中的字符串类型有 CHAR、VARCHAR、TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT、ENUM、SET 等。表 4-11 列出了 MySQL 中的字符串数据类型，括号中的 M 表示可以为其指定长度，L 表示实际字符串长度。

表 4-11　MySQL 中的字符串数据类型

类型名称	说明	存储需求
CHAR（M）	固定长度非二进制字符串	M 字节，$1 \leqslant M \leqslant 255$
VARCHAR（M）	可变长度非二进制字符串	L+1 字节，且 $L \leqslant M$ 和 $1 \leqslant M \leqslant 255$
TINYTEXT	非常小的非二进制字符串	L+1 字节，且 $L < 2^8$
TEXT	小的非二进制字符串	L+2 字节，且 $L < 2^{16}$
MEDIUMTEXT	中等大小的非二进制字符串	L+3 字节，且 $L < 2^{24}$
LONGTEXT	大的非二进制字符串	L+4 字节，且 $L < 2^{32}$
ENUM	枚举类型，只能有一个枚举字符串值	1 或 2 个字节，取决于枚举值的数目（最大值为 65 535）
SET	一个设置，字符串对象可以有零个或多个 SET 成员	1、2、3、4 或 8 个字节，取决于集合成员的数量（最多 64 个成员）

1.CHAR 和 VARCHAR 类型

CHAR（M）为固定长度字符串，M 表示列的长度，在定义时指定字符串列长，范围

是 0 ~ 255 个字符。保存时，在右侧填充空格以达到指定的长度，例如，CHAR（4）定义了一个固定长度的字符串列，包含的字符个数最大为 4。当检索到 CHAR 值时，尾部的空格将被删除。

笔记

VARCHAR（*M*）是长度可变的字符串，*M* 表示最大列的长度，*M* 的范围是 0 ~ 65535。VARCHAR 的最大实际长度由最长的行的大小和使用的字符集确定，而实际占用的空间为字符串的实际长度加 1。例如，VARCHAR（50）定义了一个最大长度为 50 的字符串，如果插入的字符串只有 10 个字符，则实际存储的字符串为 10 个字符和一个结束字符。VARCHAR 在值保存和检索时尾部的空格仍保留。

将不同的字符串保存到 CHAR（4）和 VARCHAR（4）时 CHAR 和 VARCHAR 之间的差别如表 4-12 所示。

表 4-12　CHAR 和 VARCHAR 的差别

插入值	CHAR（4）	存储需求	VARCHAR（4）	存储需求
''	' '	4 字节	''	1 字节
'ab'	'ab '	4 字节	'ab'	3 字节
'abc'	'abc '	4 字节	'abc'	4 字节
'abcd'	'abcd'	4 字节	'abcd'	5 字节
'abcdef'	'abcd'	4 字节	'abcd'	5 字节

从对比结果可以看到，CHAR（4）定义了固定长度为 4 的列，无论存入的数据长度为多少，所占用的空间均为 4 个字节；VARCHAR（4）定义的列所占的字节数为实际长度加 1。

2.TEXT 类型

TEXT 列保存非二进制字符串，如文章内容、评论等。当保存或查询 TEXT 列的值时，不删除尾部空格。TEXT 类型分为 4 种：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。

不同的 TEXT 类型的存储空间和数据长度不同，TINYTEXT 长度为 255（2^8–1）字符、TEXT 长度为 65 535（2^{16}–1）字符、MEDIUMTEXT 长度为 16 777 215（2^{24}–1）字符、LONGTEXT 长度为 4 294 967 295（2^{32}–1）或 4GB 字符。

3.ENUM 类型

ENUM 是一个字符串对象，值为表创建时规定枚举的一列值。其语法格式如下：

```
<字段名> ENUM('值 1','值 2',...,'值 n')
```

“字段名”指将要定义的字段，“值 n”指枚举列表中的第 n 个值。

ENUM 类型的字段在取值时，可以在指定的枚举列表中获取，而且一次只能获取一个。如果创建的成员中有空格，尾部的空格将自动被删除。每个枚举值均有一个索引值，在内部用整数表示，所允许的成员索引值从 1 开始编号，MySQL 存储的就是这个索引编号，枚举最多可以有 65 535 个元素。例如，定义 ENUM 类型的列（'first', 'second', 'third'），该列可以取的值和每个值的索引如表 4-13 所示。

笔记

表 4-13　ENUM 类型列的值与索引

值	索引
NULL	NULL
''	0
first	1
second	2
third	3

ENUM 值依照列索引顺序排列，并且空字符串排在非空字符串前，NULL 值排在其他所有枚举值前。

提示：

ENUM 列总有一个默认值。如果将 ENUM 列声明为 NULL，NULL 值则为该列的一个有效值，并且默认值为 NULL。如果 ENUM 列被声明为 NOT NULL，其默认值为允许的值列表的第一个元素。

4.SET 类型

SET 是一个字符串对象，可以有零个或多个值，SET 列最多可以有 64 个成员，值为表创建时规定的一列值。指定包括多个 SET 成员的 SET 列值时，各成员之间用逗号‘,’隔开，语法格式如下：

```
< 字段名 > SET( ' 值 1', ' 值 2', ... , ' 值 n' )
```

与 ENUM 类型相同，SET 值在内部用整数表示，列表中每个值都有一个索引编号。当创建表时，SET 成员值的尾部空格将自动删除。与 ENUM 类型不同的是，ENUM 类型的字段只能从定义的列值中选择一个值插入，而 SET 类型的列可从定义的列值中选择多个字符的联合。

提示：

如果插入 SET 字段中的列值有重复，则 MySQL 会自动删除重复的值。插入 SET 字段值的顺序并不重要，MySQL 会在存入数据库时，按照定义的顺序显示；如果插入了不正确的值，默认情况下，MySQL 将忽视这些值，并给出警告。

4.1.5 二进制类型

MySQL 支持两类字符型数据：文本字符串和二进制字符串，二进制字符串类型有时候也直接被称为“二进制类型”。MySQL 中的二进制类型有 BIT、BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。MySQL 中的二进制数据类型如表 4-14 所示，括号中的 *M* 表示可以为其指定的长度，*L* 表示实际的字符串长度。

表 4-14　MySQL 中的二进制数据类型

类型名称	说明	存储需求
BIT（*M*）	位字段类型	大约（*M*+7）/8 字节
BINARY（*M*）	固定长度二进制字符串	*M* 字节
VARBINARY（*M*）	可变长度二进制字符串	*L*+1 字节，且 *L* ≤ M
TINYBLOB	非常小的 BLOB	*L*+1 字节，且 $L < 2^8$
BLOB	小的 BLOB	*L*+2 字节，且 $L < 2^{16}$
MEDIUMBLOB	中等大小的 BLOB	*L*+3 字节，且 $L < 2^{24}$
LONGBLOB	非常大的 BLOB	*L*+4 字节，且 $L < 2^{32}$

笔记

1.BIT 类型

BIT（*M*）是位字段类型，*M* 表示每个值的位数，范围为 1 ~ 64。如果 *M* 被省略，默认值为 1。如果为 BIT（*M*）列分配的值的长度小于 *M* 位，在值的左边用 0 填充，例如，为 BIT（6）列分配一个值 '101'，其效果与分配 '000101' 相同。

BIT 数据类型可以用来保存位字段值，例如，以二进制的形式保存数据 13，13 的二进制形式为 1101，在这里需要位数至少为 4 位的 BIT 类型，即可以定义列类型为 BIT（4）。大于二进制 1111 的数据是不能插入 BIT（4）类型的字段中的。

提示：

默认情况下，MySQL 不可以插入超出该列允许范围的值，因而插入数据时要确保插入的值在指定的范围内。

2.BINARY 和 VARBINARY 类型

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串。使用的语法格式如下：

```
<字段名> BINARY(M) 或者 VARBINARY(M)
```

或者

```
VARBINARY(M)
```

BINARY 类型的长度是固定的，指定长度后，不足最大长度的，须在它们右边填充“\0”补齐，以达到指定长度。例如，指定列数据类型为 BINARY（3），当插入 a 时，存储的内容实际为“a\0\0”，当插入 ab 时，实际存储的内容为“ab\0”，无论存储的内容是否达到指定的长度，存储空间均为指定的值 *M*。

VARBINARY 类型的长度是可变的，指定好长度之后，其长度可以在 0 到最大值之间。例如，指定数据类型为 VARBINARY（20），如果插入的值长度只有 10，则实际存储空间为 10 加 1，实际占用的空间为字符串的实际长度加 1。

3.BLOB 类型

BLOB 是一个二进制的对象，用来存储可变数量的数据。BLOB 类型分为 4 种：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB，它们可容纳值的最大长度不同，如表 4-15 所示。

笔记

表 4-15　MySQL 中的 BLOB 数据类型

数据类型	存储范围
TINYBLOB	最大长度为 255（2^8–1）字节
BLOB	最大长度为 65 535（2^{16}–1）字节
MEDIUMBLOB	最大长度为 16 777 215（2^{24}–1）字节
LONGBLOB	最大长度为 4 294 967 295（2^{32}–1）或 4GB 字节

BLOB 列存储的是二进制字符串，TEXT 列存储的是非二进制字符串。BLOB 列是字符集，对基于列值字节的数值进行排序和比较；TEXT 列有一个字符集，根据字符集对值进行排序和比较。

不同的数据类型有不同的取值范围，并且需要不同的存储空间。因此应根据实际需要选择最合适的数据类型，这样有利于提高查询的效率和节省存储空间。

4.2 创建数据表

创建与查看数据表

创建数据库之后，接下来就要在数据库中创建数据表。所谓创建数据表，指的是在已经创建的数据库中建立新表。创建数据表的过程是规定数据列的属性的过程，同时也是实施数据完整性（包括实体完整性、引用完整性和域完整性）约束的过程。

在 MySQL 中，使用 CREATE TABLE 语句创建表，语法格式为

```
CREATE TABLE < 表名 >
(< 列名 1>< 类型 1>[AUTO_INCREMENT][ 列级约束条件 ],
< 列名 2>< 类型 2>[AUTO_INCREMENT][ 列级约束条件 ],
[...],
< 列名 n>< 类型 n>[AUTO_INCREMENT][ 列级约束条件 ],
[ 表级约束条件 ]
);
```

CREATE TABLE 命令语法比较多，具体含义如下。

（1）CREATE TABLE：用于创建给定名称的表，用户必须拥有表 CREATE 的权限。

（2）< 表名 >：指定要创建数据表的名称，在 CREATE TABLE 之后给出，必须符合标识符命名规则。表名称若被指定为 db_name.tbl_name，则可以直接在 db_name 数据库中创建表，无论 db_name 是否是当前数据库，都可以通过这种方式创建。若在当前数据库中创建表时省略 db-name，表被默认创建到当前数据库中。如果使用加引号的识别名，则应对数据库和表名称分别加引号。例如，'mydb'.'mytbl' 是合法的，但 'mydb.mytbl' 不合法。

（3）< 列名 >：指定表中各个数据列的名称，名称必须符合标识符命名规则。

（4）< 类型 >：指定表中各个数据列的数据类型，类型的选择一定要符合实际数据的使用需求。

（5）[AUTO_INCREMENT]：设置字段自动增长属性，顺序从 1 开始递增。只有数据类型为整型的列才能设置此属性。当插入 NULL 值或 0 到一个 AUTO_INCREMENT 列时，列被设置为 value+1，在这里 value 是此前表中该列的最大值。每个表只能有一个 AUTO_INCREMENT 列，并且它必须能被索引。

（6）[列级约束条件]：是指为每个数据列建立的约束条件，可以在列定义时声明，也可以在列定义后声明。比如可能的空值说明、完整性约束或表索引组成等。

（7）[表级约束条件]：是指对多个数据列建立的约束条件，只能在列定义之后声明。在实际开发中，列级约束使用更频繁。除此之外，在所有约束中，并不是每种约束都存在表级或列级约束，其中，非空约束和默认约束就不存在表级约束，它们只有列级约束，而主键约束、唯一约束、外键约束都存在表级约束和列级约束。

使用 CREATE TABLE 创建表时，如果创建多个列，则要用逗号隔开，需要特别注意的是创建的表的名称不区分大小写，且不能使用 SQL 语言中的关键字，如 DROP、ALTER、INSERT 等。

数据表属于数据库，在创建数据表之前，须使用语句 USE< 数据库 > 指定操作在哪个数据库中进行，如果没有选择数据库，就会抛出“No database selected”错误。

【实例 4-1】在数据库“EduSys”中，创建学生信息表“Student”，表中各字段的名称和数据类型如表 4-1 所示。首先，指定当前使用数据库，输入的 SQL 语句如下：

```
mysql> USE EduSys;
```

语句执行结果为

```
Database changed
```

当前数据库跳转成功。接下来，使用 CREATE TABLE 命令创建数据表“Student”，该表中暂时没有添加任何约束条件，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8),
    -> Sname varchar(20),
    -> Sex char(2),
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11)
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

语句执行成功后，一个名称为“Student”的数据表就创建好了。

4.3 查看数据表

数据表创建完成后，可以使用 SHOW TABLES 语句查看数据表是否创建成功。

【实例 4-2】查看数据库“EduSys”中的数据表，输入的 SQL 语句如下：

```
mysql> SHOW TABLES;
```

笔记

语句执行结果为

```
+----------------------------+
| Tables_in_edusys           |
+----------------------------+
| student                    |
+----------------------------+
1 row in set (0.00 sec)
```

以上结果表明数据表已创建成功。使用 SHOW TABLE STATUS 语句可以进一步查看数据表的状态信息，语法格式如下：

```
SHOW TABLE STATUS [FROM 数据库名 ] [LIKE 匹配模式 ];
```

【实例 4-3】查看数据库“ EduSys”中的表名称中包含“ stu”的数据表的状态。输入的 SQL 语句如下：

```
SHOW TABLE STATUS FROM EduSys LIKE '%stu%'\G
```

语句执行结果为

```
*************************** 1. row ***************************
           Name: student
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 17
 Avg_row_length: 963
    Data_length: 16384
Max_data_length: 0
   Index_length: 32768
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2021-06-06 15:38:32
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_unicode_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

上述结果中显示了数据表“Student”的相关信息，结果中各字段含义如表 4-16 所示。

表 4-16　数据表的相关信息

字段名称	具体含义
Name	表名称
Engine	表的存储引擎
Version	版本

（续表）

笔记

字段名称	具体含义
Row_format	行存储格式，MyISAM 引擎可能是 Dynamic、Fixed 或 Compressed
Rows	表中的行数
Avg_row_length	平均每行包括的字节数
Data_length	整个表的数据量（单位：字节）
Max_data_length	表可以容纳的最大数据量
Index_length	索引占用磁盘的空间大小
Data_free	标识已分配但现在未使用的空间，且包含已删除行的空间
Auto_increment	下一个 Auto_increment 的值
Create_time	表的创建时间
Update_time	表的最近更新时间
Check_time	使用 check table 或 myisamchk 工具检查表的最近时间
Collation	表的默认字符集和字符排序规则
Checksum	如果启用，则对整个表的内容计算时的校验和
Create_options	指表创建时的其他所有选项
Comment	其他额外信息

数据表创建好之后，可以使用 SQL 语句查看表结构的定义，确定数据表包含的字段、字段类型、宽度等是否正确。在 MySQL 中，查看表结构可以使用 DESCRIBE/DESC 或 SHOW CREATE TABLE 语句实现。DESCRIBE/DESC 语句可以查看表的字段信息，包括字段名、字段数据类型、是否为主键、是否有默认值等，语法规则如下：

```
DESCRIBE < 表名 >;
```

或简写为：

```
DESC < 表名 >;
```

【实例 4-4】使用 DESCRIBE 查看表“Student”的结构，输入的 SQL 语句如下：

```
mysql> DESCRIBE Student;
```

语句执行结果为

```
+-------------+-----------------+----------+-------+------------+--------+
| Field       | Type            | Null     | Key   | Default    | Extra  |
+-------------+-----------------+----------+-------+------------+--------+
| Sno         | char(8)         | YES      |       | NULL       |        |
| Sname       | varchar(20)     | YES      |       | NULL       |        |
| Sex         | char(2)         | YES      |       | NULL       |        |
| Native      | varchar(30)     | YES      |       | NULL       |        |
| Birthday    | datetime        | YES      |       | NULL       |        |
| Major       | varchar(20)     | YES      |       | NULL       |        |
| Classno     | char(4)         | YES      |       | NULL       |        |
| Tel         | char(11)        | YES      |       | NULL       |        |
+-------------+-----------------+----------+-------+------------+--------+
8 rows in set (0.01 sec)
```

笔记

结果中各个字段的含义如下。

（1）Field：表示表中各个字段的名称。

（2）Type：表示表中各个字段的数据类型。

（3）Null：表示该列是否可以为空值，若值为“Yes”表示可以为空值，为“No”表示不允许为空值。

（4）Key：表示该列是否已设置索引。若值为“PRI”表示该列是表主键的一部分，值为“UNI”表示该列是 UNIQUE 唯一索引的一部分，值为“MUL”表示在列中某个给定值允许出现多次。

（5）Default：表示该列是否有默认值，如果有，取值为设置的默认值，否则显示值为“NULL”。

（6）Extra：表示可获取的与给定列有关的附加信息，如 AUTO_INCREMENT 等。

使用 SHOW CREATE TABLE 语句可以用来显示创建表时的 CREATE TABLE 语句，语法格式如下：

```
SHOW CREATE TABLE < 表名 >\G
```

【实例 4-5】使用 SHOW CREATE TABLE 查看表“Student”的详细信息，输入的 SQL 语句如下：

```
mysql> SHOW CREATE TABLE Student\G
```

语句执行结果为

```
*************************** 1. row ***************************
       Table: student
Create Table: CREATE TABLE `student` (
  `Sno` char(8) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Sname` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Sex` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Native` varchar(30) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Birthday` datetime DEFAULT NULL,
  `Major` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Classno` char(4) COLLATE utf8_unicode_ci DEFAULT NULL,
  `Tel` char(11) COLLATE utf8_unicode_ci DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
1 row in set (0.00 sec)
```

使用 SHOW CREATE TABLE 语句不仅可以查看创建表时的详细语句，而且可以查看存储引擎和字符编码。如果不加“\G”参数，显示的结果可能非常混乱，加上“\G”参数之后，可使显示的结果更加直观，易于查看。

4.4 修改数据表

为实现数据库中表的规范化设计，有时候需要对之前已经创建的数据表进行结构修改或者调整。修改表指的是修改数据库中已经存在的数据表结构。在 MySQL 中可以使用

ALTER TABLE 语句来改变原有表的结构，常用的修改表的操作有修改表名、修改字段名称、修改字段数据类型、增加和删除字段、调整字段的排列位置、更改表的存储引擎、添加和删除约束条件、重命名表等。完整语法格式如下：

```
ALTER TABLE < 表名 > [ 修改选项 ]
```

修改选项的语法格式如下：

```
{ADD COLUMN < 列名 > < 类型 >
|CHANGE COLUMN < 旧列名 > < 新列名 > < 新列类型 >
|ALTER COLUMN < 列名 >{SET DEFAULT < 默认值 > | DROP DEFAULT}
|MODIFY COLUMN < 列名 > < 类型 >
|DROP COLUMN < 列名 >
|RENAME TO < 新表名 >}
```

下面将分情况介绍修改数据表的操作。

4.4.1 添加字段

随着业务的变化，可能需要在已经存在的表中添加新的字段，一个完整的字段包括字段名、数据类型、完整性约束。添加字段的语法格式如下：

```
ALTER TABLE< 表名 >
ADD < 新字段名 > < 数据类型 >[ 约束条件 ][FIRST|AFTER 已存在的字段名 ];
```

语法中参数的具体含义如下。

（1）表名：指定修改的数据表的名称。

（2）新字段名：指需要添加的字段的名称。

（3）数据类型：指新添加的字段的数据类型。

（4）[约束条件]：新添加的字段需满足的约束条件。

（5）[FIRST]：为可选参数，其作用是将新添加的字段设置为表的第一个字段。

（6）[AFTER] ：为可选参数，其作用是将新添加的字段添加到指定的已存在的字段名的后面。

【实例 4-6】修改表“Student”结构，增加“ID”序号字段，类型为 CHAR（6），并作为表中的第一个字段，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> ADD COLUMN ID CHAR(6) FIRST;
```

语句执行结果为

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

笔记

添加字段

笔记

语句执行结果为

```
+-------------+-----------------+----------+-------+------------+--------+
| Field       | Type            | Null     | Key   | Default    | Extra  |
+-------------+-----------------+----------+-------+------------+--------+
| ID          | char(6)         | YES      |       | NULL       |        |
| Sno         | char(8)         | YES      |       | NULL       |        |
| Sname       | varchar(20)     | YES      |       | NULL       |        |
| Sex         | char(2)         | YES      |       | NULL       |        |
| Native      | varchar(30)     | YES      |       | NULL       |        |
| Birthday    | datetime        | YES      |       | NULL       |        |
| Major       | varchar(20)     | YES      |       | NULL       |        |
| Classno     | char(4)         | YES      |       | NULL       |        |
| Tel         | char(11)        | YES      |       | NULL       |        |
+-------------+-----------------+----------+-------+------------+--------+
9 rows in set (0.01 sec)
```

从结果中可以看出，“ID”列已经添加成功，并且处于表“Student”中第一列。

【实例 4-7】修改表“Student”结构，在“Native”字段后添加一个字段“Code”，类型为 CHAR（6），输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> ADD COLUMN Code CHAR(6) AFTER Native;
```

语句执行结果为

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

语句执行结果为

```
+-------------+-----------------+----------+-------+------------+--------+
| Field       | Type            | Null     | Key   | Default    | Extra  |
+-------------+-----------------+----------+-------+------------+--------+
| ID          | char(6)         | YES      |       | NULL       |        |
| Sno         | char(8)         | YES      |       | NULL       |        |
| Sname       | varchar(20)     | YES      |       | NULL       |        |
| Sex         | char(2)         | YES      |       | NULL       |        |
| Native      | varchar(30)     | YES      |       | NULL       |        |
| Code        | char(6)         | YES      |       | NULL       |        |
| Birthday    | datetime        | YES      |       | NULL       |        |
| Major       | varchar(20)     | YES      |       | NULL       |        |
| Classno     | char(4)         | YES      |       | NULL       |        |
| Tel         | char(11)        | YES      |       | NULL       |        |
+-------------+-----------------+----------+-------+------------+--------+
10 rows in set (0.01 sec)
```

可以看到，表“Student”中增加了一个名称为“Code”的字段，其位置在指定的“Native”字段后面，说明字段添加成功。

笔记

提示：

“FIRST | AFTER 已存在的字段名”用于指定新增字段在表中的位置，如果 SQL 语句中没有这两个参数，则默认将新添加的字段设置为数据表的最后一列。

4.4.2 修改字段数据类型

修改字段的数据类型就是把字段的数据类型转换成另一种数据类型。在 MySQL 中修改字段数据类型的语法规则如下：

修改现有字段数据类型

```
ALTER TABLE < 表名 >
MODIFY < 字段名 > < 数据类型 >;
```

其中，表名指要修改数据类型的字段所在表的名称，字段名指需要修改的字段，数据类型指修改后字段的新数据类型。

【实例 4-8】修改表“Student”结构，将“Sname”字段的数据类型由 VARCHAR（20）修改成 VARCHAR（40），输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> MODIFY Sname VARCHAR(40);
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

语句执行结果为

```
+-------------+-----------------+-------+-------+------------+--------+
| Field       | Type            | Null  | Key   | Default    | Extra  |
+-------------+-----------------+-------+-------+------------+--------+
| ID          | char(6)         | YES   |       | NULL       |        |
| Sno         | char(8)         | YES   |       | NULL       |        |
| Sname       | varchar(40)     | YES   |       | NULL       |        |
| Sex         | char(2)         | YES   |       | NULL       |        |
| Native      | varchar(30)     | YES   |       | NULL       |        |
| Code        | char(6)         | YES   |       | NULL       |        |
| Birthday    | datetime        | YES   |       | NULL       |        |
| Major       | varchar(20)     | YES   |       | NULL       |        |
| Classno     | char(4)         | YES   |       | NULL       |        |
| Tel         | char(11)        | YES   |       | NULL       |        |
+-------------+-----------------+-------+-------+------------+--------+
10 rows in set (0.01 sec)
```

笔记

语句执行后，发现表“Student”中“Sname”字段的数据类型已经修改成 VARCHAR(40), 说明修改成功。

注意：

若表中该列所保存数据的数据类型与将要修改的列的新数据类型冲突，则会发生错误。比如，若将原来 Char 类型的列修改成 Int 类型，而原来已经保存有字符型数据，则无法修改。

4.4.3 调整字段排列顺序

调整字段顺序及删除字段

如果要调整字段的排列顺序，则使用的语法规则如下：

```
ALTER TABLE < 表名 >
MODIFY [COLUMN] 字段名 1 数据类型 [ 字段属性 ]
[FIRST | AFTER 字段名 2];
```

其中，字段名 1 指表中需要调整顺序的字段的名称；字段名 2 指调整到其位置之后的字段的名称。

【实例 4-9】修改表“Student”结构，将“Birthday”字段调整到“Sex”字段之后，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> MODIFY Birthday datetime AFTER Sex;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

语句执行结果为

```
+-------------+-------------------+-------+-------+-----------+--------+
| Field       | Type              | Null  | Key   | Default   | Extra  |
+-------------+-------------------+-------+-------+-----------+--------+
| ID          | char(6)           | YES   |       | NULL      |        |
| Sno         | char(8)           | YES   |       | NULL      |        |
| Sname       | varchar(40)       | YES   |       | NULL      |        |
| Sex         | char(2)           | YES   |       | NULL      |        |
| Birthday    | datetime          | YES   |       | NULL      |        |
| Native      | varchar(30)       | YES   |       | NULL      |        |
| Code        | char(6)           | YES   |       | NULL      |        |
| Major       | varchar(20)       | YES   |       | NULL      |        |
| Classno     | char(4)           | YES   |       | NULL      |        |
| Tel         | char(11)          | YES   |       | NULL      |        |
+-------------+-------------------+-------+-------+-----------+--------+
10 rows in set (0.01 sec)
```

语句执行后，发现表“Student”中的“Birthday”字段已调整到“Sex”字段后面，说明修改成功。

笔记

4.4.4 删除字段

删除字段是将数据表中的某个字段从表中移除，语法格式如下：

```
ALTER TABLE < 表名 >
DROP < 字段名 >;
```

其中，字段名指需要从表中删除的字段的名称。

【实例 4-10】修改表“Student”结构，删除“ID”字段，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> DROP ID;
```

语句执行结果为

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

语句执行结果为

```
+-------------+-----------------+------+-----+---------+--------+
| Field       | Type            | Null | Key | Default | Extra  |
+-------------+-----------------+------+-----+---------+--------+
| Sno         | char(8)         | YES  |     | NULL    |        |
| Sname       | varchar(40)     | YES  |     | NULL    |        |
| Sex         | char(2)         | YES  |     | NULL    |        |
| Birthday    | datetime        | YES  |     | NULL    |        |
| Native      | varchar(30)     | YES  |     | NULL    |        |
| Code        | char(6)         | YES  |     | NULL    |        |
| Major       | varchar(20)     | YES  |     | NULL    |        |
| Classno     | char(4)         | YES  |     | NULL    |        |
| Tel         | char(11)        | YES  |     | NULL    |        |
+-------------+-----------------+------+-----+---------+--------+
9 rows in set (0.01 sec)
```

语句执行后，发现表“Student”中的“ID”字段已被成功删除。

4.4.5 修改字段名称

修改字段名称是指将原有字段重命名，语法规则如下：

修改字段名及表名

笔记

```
ALTER TABLE < 表名 >
CHANGE < 旧字段名 > < 新字段名 > < 新数据类型 >;
```

其中，旧字段名指修改前的字段名；新字段名指修改后的字段名；新数据类型指修改后的数据类型。

【实例 4-11】修改表“Student”结构，将“Code”字段名称改为“ZipCode”，数据类型不变，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> CHANGE Code ZipCode CHAR(6);
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

利用 DESC 命令查看当前表的结构，输入的 SQL 语句如下：

```
mysql> DESC Student;
```

语句执行结果为

```
+-------------+-----------------+-------+-------+-----------+--------+
| Field       | Type            | Null  | Key   | Default   | Extra  |
+-------------+-----------------+-------+-------+-----------+--------+
| Sno         | char(8)         | YES   |       | NULL      |        |
| Sname       | varchar(40)     | YES   |       | NULL      |        |
| Sex         | char(2)         | YES   |       | NULL      |        |
| Birthday    | datetime        | YES   |       | NULL      |        |
| Native      | varchar(30)     | YES   |       | NULL      |        |
| ZipCode     | char(6)         | YES   |       | NULL      |        |
| Major       | varchar(20)     | YES   |       | NULL      |        |
| Classno     | char(4)         | YES   |       | NULL      |        |
| Tel         | char(11)        | YES   |       | NULL      |        |
+-------------+-----------------+-------+-------+-----------+--------+
9 rows in set (0.01 sec)
```

语句执行后，发现表“Student”中“Code”字段名称已经被修改为“ZipCode”，修改成功。

说明：

使用 CHANGE 语句时，如果不需要修改字段的数据类型，则可将新数据类型设置成与原来一样，但数据类型不能为空。CHANGE 也可以只修改数据类型，实现和 MODIFY 同样的效果，方法是将“新字段名”和“旧字段名”设置为相同的名称，只改变“新数据类型”。

提示：

由于不同类型的数据在机器中的存储方式及长度并不相同，修改数据类型可能会影响数据表中已有的数据记录。因此，当数据表中已经有数据时，不要轻易修改数据类型。

笔记

4.4.6 修改表名称

在 MySQL 中使用 ALTER TABLE 语句可以实现表名的修改，语法规则如下：

```
ALTER TABLE < 旧表名 >
RENAME [TO] < 新表名 >;
```

其中，TO 为可选参数，使用与否均不影响结果。

【实例 4-12】修改表“Student”的名称为“Student_New”，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> RENAME Student_New;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

利用“SHOW TABLES”命令查看表，输入的 SQL 语句如下：

```
mysql> SHOW TABLES;
```

语句执行结果为

```
+------------------------+
| Tables_in_edusys       |
+------------------------+
| student_new            |
+------------------------+
1 row in set (0.00 sec)
```

用户在修改表名称时可以进一步使用 DESC 命令查看修改后的表结构，会发现修改表名并不影响表的结构，因此修改名称后的表和修改名称前的表的结构是相同的。

4.5 删除数据表

对于不再需要的数据表，可以将其从数据库中删除。在删除表的同时，表的结构和表中所有的数据都会被删除，包括表的描述、完整性约束、索引及与表相关的权限等，因此在删除数据表之前最好先备份，以免造成无法挽回的损失。

MySQL 数据库中使用 DROP TABLE 语句删除一个或多个数据表，语法格式如下：

```
DROP TABLE [IF EXISTS] 表名 1 [, 表名 2, 表名 3, ...];
```

删除数据表

笔记

语法说明如下：

（1）表名 1，表名 2，表名 3 表示要被删除的数据表的名称。DROP TABLE 可以同时删除多个表，只要将表名依次写在后面，相互之间用逗号隔开。

（2）IF EXISTS 用于在删除数据表之前判断该表是否存在。如果不加 IF EXISTS，当数据表不存在时 MySQL 将提示错误，中断 SQL 语句的执行；加上之后，当数据表不存在时，SQL 语句仍可顺利执行，但是会发出警告（warning）。

注意：

用户必须拥有执行 DROP TABLE 命令的权限，否则数据表不会被删除。表被删除时，用户在该表上的权限不会自动删除。

【实例 4-13】删除数据表“Student_New”，输入的 SQL 语句如下：

```
mysql> DROP TABLE Student_New;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

利用 SHOW TABLES 命令查看表，输入的 SQL 语句如下：

```
mysql> SHOW TABLES;
```

语句执行结果为

```
Empty set (0.00 sec)
```

执行结果可以看到，此时“EduSys”数据库已经不存在任何数据表，输出结果为“Empty set”，此时删除“Student_New”数据表操作成功。

4.6 数据完整性

4.6.1 数据完整性概述

数据完整性

复杂的数据类型可以满足不同类型的数据需求，但实际应用中的数据还有着更多的要求，比如年龄不小于 0、性别取值为“男”或“女”、表间关联字段要保持一致和完整等。然而，数据库中的数据是从外界输入的，实际操作无法保证插入和修改的数据都符合要求，不符合要求的操作有可能会破坏数据的完整性，对数据库的可靠性和运行能力造成威胁。因此数据库必须对数据表和列有所限制和规范，使用一系列的方法来维护数据完整性就成了数据库系统尤其是多用户的关系数据库系统首要关注的问题。

数据完整性（data integrity）是指数据库中数据的精确性（accuracy）、一致性（consistency）和可靠性（reliability），用以防止数据库中存在不符合语义规定的数据，防止因错误信息的输入或输出造成无效操作，解决表内数据矛盾、表间数据矛盾及关联不一致的问题。数据完整性分为实体完整性（entity integrity）、域完整性（domain integrity）、参照

完整性（referential integrity）3 类。

笔记

1. 域完整性

域完整性又称列完整性，指数据表中的列必须满足某种特定的数据类型或约束。定义域完整性的方法有限制类型（通过设定列的数据类型）、格式或可能值的范围（通过 FOREIGN KEY 约束、DEFAULT 定义、NOT NULL 定义和规则等）。如性别只能是“男”或“女”。

2. 实体完整性

实体完整性又称行完整性，要求表中的所有行都有一个唯一标识符，保证表中所有的行唯一。唯一标识符可能是一列，也可能是几个列的组合，称为主键。表中的主键在所有行上必须取唯一值。定义实体完整性的方法有索引、UNIQUE 约束、PRIMARY KEY 约束或 IDENTITY 属性。如：Student 表中学号 Sno 的取值必须唯一，它唯一标识了相应记录的学生，可以设置为主键。

3. 参照完整性

参照完整性是指两个表的主关键字和外部关键字的数据应对应一致，它确保了有主关键字的表中对应其他表的外关键字的行存在。在输入、更改或删除记录时，参照完整性保持表之间已定义的关系，确保键值在所有表中一致。这样的一致性要求可以确保不会引用不存在的值，如果键值更改了，那么在整个数据库中，对该键值的所有引用要一致更改，以防止数据丢失或无意义的数据在数据库中扩散。例如，学生学习课程的课程号必须是有效的课程号，学生选课表（StuCourse 表）的外键 Cno（课程编号）将参考课程信息表（Course 表）中主键 Cno（课程编号）以实现参照完整性。

域完整性、实体完整性及参照完整性分别在列、行、表上实施。数据完整性任何时候都可以实施，但对已有数据的表实施数据完整性时，系统要先检查表中的数据是否满足所实施的完整性，只有表中的数据满足了所实施的完整性，数据完整性才能实施成功。

4.6.2 完整性约束条件

数据完整性非常重要，那么如何维护数据的完整性呢？在 MySQL 中，约束是维护数据完整性的强有力手段，可以确保有效的数据插入表中，并维护表和表之间的特定关系。MySQL 所提供的约束和数据完整性的关系如表 4-17 所示。

表 4-17　MySQL 约束类型及相应的完整性类型

约束类型	描述	完整性类型
非空约束（NOT NULL）	指定某个列不可以取空值	域完整性
默认约束（DEFAULT）	添加数据时若未定义取值，则为列插入默认值	域完整性
主键约束（PRIMARY KEY）	每行的唯一标识符，不允许空值	实体完整性
唯一性约束（UNIQUE）	防止冗余值，保证值的唯一性，允许空值	实体完整性
外键约束（FOREIGN KEY）	定义一列或几列的值与另一个表的主键值相匹配	参照完整性

定义约束可以分为两种情况来完成：利用 CREATE TABLE 命令创建表过程中定义约束，即从无到有创建约束，在创建表的同时创建约束；利用 ALTER TABLE 命令修改表，

笔记

为表定义约束，即在已有的表上添加约束，或对已有的约束进行修改。

1. 主键约束

主键约束（PRIMARY KEY）是指一个列或多列的组合，其值能唯一地标识表中的每一行记录，这样的一列或多列称为表的主键，其中由多列组合的主键称为复合主键，通过它可以强化表的实体完整性。主键应遵守下面的规则。

主键约束

第一，每个表只能定义一个主键。

第二，主键值必须唯一标识表中的每一行，且不能为 NULL，即表中不可能存在两行数据有相同的主键值，这是唯一性原则。

第三，一个列名只能在复合主键列表中出现一次。

第四，复合主键不能包含不必要的多余列，须满足最小化原则。

1）创建表时利用列级约束设置主键约束

在 CREATE TABLE 语句中，通过 PRIMARY KEY 关键字指定主键，在定义列的同时指定主键，语法规则如下：

```
< 字段名 >< 数据类型 > Primary Key;
```

【实例 4-14】在数据库“EduSys”中，创建学生信息表“Student”，表中各字段的名称和数据类型如表 4-1 所示，其主键为“Sno”字段，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8) Primary Key,
    -> Sname varchar(20),
    -> Sex char(2),
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11)
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

2）创建表时利用表级约束设置主键约束

如果是在定义完所有列之后，再指定主键，那此时使用的是表级约束。表级约束可以在创建表时设置复合主键，即主键由多个字段联合组成，语法格式如下：

```
[CONSTRAINT < 约束名 >] Primary Key [ 字段 1, 字段 2,..., 字段 n]
```

说明如下。

（1）[CONSTRAINT< 约束名 >]：CONSTRAINT 是约束关键字，< 约束名 > 是为所创建的约束取的名称，该项可以省略。若省略，则系统会自动为其赋予一个名称。

（2）Primary Key：表明定义的是主键约束。

（3）[字段 1, 字段 2,..., 字段 n] ：设置为主键的字段名，可以是单一字段，也可以是多个字段。不论是单一字段定义的单一主键还是多个字段定义的复合主键，一张表的主键约束只能有一个。

笔记

【实例 4-15】在数据库“ EduSys”中，创建学生选课表“ StuCourse”，表中各字段的名称和数据类型如表 4-5 所示，其主键为“ Sno”字段和“ Cno”字段组合构成的复合主键，输入的 SQL 语句如下：

```
mysql> CREATE TABLE StuCourse
    -> (
    -> Sno char(8),
    -> Cno char(5),
    -> Score tinyint,
    -> Primary Key(Sno,Cid)
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

3）修改表时为表添加主键约束

在修改数据表时添加主键约束的语法规则如下：

```
ALTER TABLE < 数据表名 > ADD PRIMARY KEY(< 列名 >);
```

【实例 4-16】在数据库“ EduSys”中，创建只包含字段名和数据类型的课程信息表“ Course”，如表 4-2 所示。然后，修改表“ Course”为其添加主键，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Course
    -> ADD PRIMARY KEY (Cno);
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

2. 非空约束

非空约束（NOT NULL）指字段的值不能为空。对于使用了非空约束的字段，如果用户在添加数据时没有指定值，数据库系统就会报错。非空约束可以通过 CREATE TABLE 语句或 ALTER TABLE 语句实现。用户可以通过在表中某个列的定义后加上关键字 NOT NULL 作为限定词，来约束该列的取值不能为空。

非空约束

1）创建表时设置非空约束

创建表时可以使用 NOT NULL 关键字设置非空约束，具体的语法规则如下：

```
< 字段名 > < 数据类型 > NOT NULL;
```

笔记

【实例 4-17】在数据库“EduSys”中，创建学生信息表“Student”，表中各字段的名称和数据类型如表 4-1 所示，要求“Sname”字段不可以为空值，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8) Primary Key,
    -> Sname varchar(20) NOT NULL,
    -> Sex char(2),
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11)
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

2）修改表时添加非空约束

修改表时设置非空约束的语法规则如下：

```
ALTER TABLE < 数据表名 >
CHANGE COLUMN < 字段名 >
< 字段名 > < 数据类型 > NOT NULL;
```

【实例 4-18】在数据库“EduSys”中，为学生信息表“Student”中的“Sex”字段添加非空约束，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> CHANGE COLUMN Sex
    -> Sex char(2) NOT NULL;
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3）删除非空约束

可以借助修改表命令，删除非空约束，语法规则如下：

```
ALTER TABLE < 数据表名 >
CHANGE COLUMN < 字段名 >
< 字段名 > < 数据类型 > NULL;
```

【实例 4-19】在数据库“EduSys”中，为学生信息表“Student”中的“Sex”字段修改约束，允许其取空值，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Student
    -> CHANGE COLUMN Sex
    -> Sex char(2) NULL;
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

笔记

3. 唯一约束

唯一约束（UNIQUE）要求该列值唯一，允许为空，但只能出现一个空值。唯一约束可以确保一列或者几列不出现重复值。

唯一约束

1）创建表时设置唯一约束

在定义完列之后，直接使用 UNIQUE 关键字指定唯一约束，语法规则如下：

```
< 字段名 >< 数据类型 > UNIQUE
```

【实例 4-20】在数据库“EduSys”中，创建学生信息表“Student”，表中各字段的名称和数据类型如表 4-1 所示，要求“Tel”字段值唯一，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8) Primary Key,
    -> Sname varchar(20) NOT NULL,
    -> Sex char(2),
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11) UNIQUE
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

提示：

UNIQUE 和 PRIMARY KEY 的区别在于：一个表可以有多个字段声明为 UNIQUE，但只能有一个字段声明为 PRIMARY KEY；声明为 PRIMAY KEY 的列不允许有空值，但是声明为 UNIQUE 的字段允许空值的存在。

2）修改表时添加唯一约束

在修改表时添加唯一约束，其语法格式为

```
ALTER TABLE < 数据表名 >
ADD CONSTRAINT < 唯一约束名 > UNIQUE(< 列名 >);
```

笔记

【实例 4-21】在数据库“EduSys”中，为课程信息表“Course”中的“Cname”字段添加唯一约束，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Course
    -> ADD CONSTRAINT Unique_Cname UNIQUE(Cname);
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3）删除唯一约束

在 MySQL 中删除唯一约束的语法格式如下：

```
ALTER TABLE < 表名 > DROP INDEX < 唯一约束名 >;
```

【实例 4-22】在数据库“EduSys”中，将课程信息表“Course”中的“Cname”字段的唯一约束删除，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Course
    -> DROP INDEX Unique_Cname;
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. 默认值约束

默认值约束（DEFAULT）用来指定某列的默认值。例如，学生中男性较多时，性别就可以设置默认值为“男”。如果插入一条新的记录时没有为这个字段赋值，那么系统会自动为这个字段赋值为“男”。

默认值约束

1）创建表时设置默认值约束

创建表时可以使用 DEFAULT 关键字设置默认值约束，具体的语法规则如下：

```
< 字段名 > < 数据类型 > DEFAULT < 默认值 >;
```

【实例 4-23】在数据库“EduSys”中，创建学生信息表“Student”，表中各字段的名称和数据类型如表 4-1 所示，要求“Sex”字段设置默认值“男”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8) Primary Key,
    -> Sname varchar(20) NOT NULL,
    -> Sex char(2) DEFAULT ' 男 ',
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11) UNIQUE
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

以上语句执行成功之后，表“ Student”中的“ Sex”字段拥有了一个默认值“男”，新插入的学生记录没有指定性别信息时，则默认都为“男”。

2）修改表时添加默认值约束

修改表时添加默认值约束的语法规则如下：

```
ALTER TABLE < 数据表名 >
CHANGE COLUMN < 字段名 >
< 字段名 > < 数据类型 > DEFAULT < 默认值 >;
```

【实例 4-24】在数据库“ EduSys”中，为课程信息表“ Course”中的“ Hours”字段添加默认值为“64”，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Course
    -> CHANGE COLUMN Hours
    -> Hours tinyint DEFAULT 64;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3）删除默认值约束

修改表时删除默认值约束的语法规则如下：

```
ALTER TABLE < 数据表名 >
CHANGE COLUMN < 字段名 >
< 字段名 > < 数据类型 > DEFAULT NULL;
```

【实例 4-25】在数据库“ EduSys”中，将课程信息表“ Course”中的“ Hours”字段的默认值删除，输入的 SQL 语句如下：

```
mysql> ALTER TABLE Course
    -> CHANGE COLUMN Hours
    -> Hours tinyint DEFAULT NULL;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

5. 外键约束

外键约束（FOREIGN KEY）用来在两个表的数据之间建立连接。一个表可以有一个或多个外键。外键可以实现数据的参照完整性，作用是保持数据的一致性、完整性。

外键关联的两个表，一个叫主表（父表），另一个叫从表（子表）。对于两个具有关联

笔记

外键约束

笔记

关系的表而言，相关联字段中主键所在的表就是主表，相关联字段中外键所在的表就是从表。外键是表的一个字段，不是本表的主键，但对应另一个表的主键。定义外键后，从表中的外键字段的取值受限于主表中的主键字段的值，从表中的外键字段要么取主表中主键字段存在的值，要么取 NULL 值。

设置 MySQL 外键约束的字段在被定义为外键时，需要遵守下列规则。

第一，父表必须已经存在于数据库中，或者是当前正在创建的表。如果是后一种情况，则父表与子表是同一个表，这样的表称为自参照表，这种结构称为自参照完整性。

第二，外键必须为关联父表中定义的主键。

第三，主键不能包含空值，但允许在外键中出现空值。也就是说，只要外键的每个非空值出现在指定的主键中，这个外键的内容就是正确的。

第四，在父表的表名后面指定列名或列名的组合。这个列或列的组合必须是父表的主键或候选键。

第五，外键中列的数目必须和父表的主键中列的数目相同。

第六，外键中列的数据类型必须和父表主键中对应列的数据类型相同。

1）创建表时设置外键约束

在数据表中创建外键须使用 FOREIGN KEY 关键字，具体的语法规则如下：

```
[CONSTRAINT < 外键约束名称 >]
FOREIGN KEY( 字段名 [, 字段名 2,...])
REFERENCES < 主表名 >( 主键列 1 [, 主键列 2,...])
```

说明如下。

（1）外键约束名称为定义的外键约束的名称，一个表中不能有相同名称的外键。

（2）字段名表示子表需要添加外键约束的字段名称。

（3）主表名即被子表外键所依赖的表的名称。

（4）主键列表示主表中定义的主键列或者列组合。

【实例 4-26】在数据库“EduSys”中，创建学生选课表“StuCourse”，表中各字段的名称和数据类型如表 4-5 所示，要求为“Sno”字段添加外键约束，与“Student”表中主键关联，输入的 SQL 语句如下：

```
mysql> CREATE TABLE StuCourse
    -> (
    -> Sno char(8),
    -> Cno char(5),
    -> Score tinyint,
    -> Primary Key(Sno,Cno),
    -> CONSTRAINT FK_Sno
    -> FOREIGN KEY(Sno)
    -> REFERENCES Student(Sno)
    -> );
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
```

以上语句执行成功之后，在学生选课表“StuCourse”上添加了名称为“FK_Sno”的外键约束，外键字段为“Sno”，其依赖于学生信息表“Student”的主键字段“Sno”。

提示：

关联指的是关系数据库中，相关表之间的联系。它是通过相同的属性或属性组来表示的。子表的外键必须关联父表的主键，且关联字段的数据类型必须匹配，如果类型不一样，则创建子表时会出现错误“ERROR 1005（HY000）：Can't create table database.tablename'（errno:150）”。

2）修改表时添加外键约束

在修改数据表时添加外键约束的语法规则为

```
ALTER TABLE < 数据表名 >
ADD CONSTRAINT < 外键约束名称 >
FOREIGN KEY(< 字段名 >)
REFERENCES < 主表名 >(< 主键列名 >)
```

【实例 4-27】在数据库“EduSys”中，修改学生选课表“StuCourse”，为“Cno”字段添加外键约束，与“Course”表主键“Cno”进行关联，输入的 SQL 语句如下：

```
mysql> ALTER TABLE StuCourse
    -> ADD CONSTRAINT FK_Cno
    -> FOREIGN KEY(Cno)
    -> REFERENCES Course(Cno);
```

语句执行结果为

```
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

以上语句执行成功之后，在学生选课表“StuCourse”上添加了名称为“FK_Cno”的外键约束，外键字段为“Cno”，其依赖于课程信息表“Course”的主键字段“Cno”。值得一提的是，主键字段和外键字段的名称可以不同，但数据类型必须相同。

利用 SHOW CREATE TABLE 命令查看表，输入语句如下：

```
mysql> SHOW CREATE TABLE StuCourse\G
```

语句执行结果为

```
*************************** 1. row ***************************
       Table: stucourse
Create Table: CREATE TABLE `stucourse` (
  `sno` char(8) COLLATE utf8_unicode_ci NOT NULL,
  `cno` char(5) COLLATE utf8_unicode_ci NOT NULL,
```

笔记

```
  `score` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`sno`,`cno`),
  KEY `fk_cno` (`cno`),
  CONSTRAINT `Fkey_sno` FOREIGN KEY (`sno`) REFERENCES `student` (`Sno`),
  CONSTRAINT `fk_cno` FOREIGN KEY (`cno`) REFERENCES `course` (`Cno`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
1 row in set (0.00 sec)
```

3）删除外键约束

数据库中定义的外键，如果不再需要，可以将其删除。外键一旦删除，就会解除主表和从表之间的关联关系，删除外键的语法格式如下：

```
ALTER TABLE < 数据表名 >
DROP FOREIGN KEY < 外键约束名称 >
```

【实例 4-28】在数据库“EduSys”中，删除学生选课表“StuCourse”中的外键约束“FK_Cno”，输入的 SQL 语句如下：

```
mysql> ALTER TABLE StuCourse
    -> DROP FOREIGN KEY FK_Cno;
```

语句执行结果为

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

语句执行成功后，“StuCourse”表中已经不存在名为“FK_Cno”的外键约束，删除成功。

任务实践

根据“学习任务”中的实践需求，利用 CREATE TABLE 语句，完成表 4-1 至表 4-6 所示的六个数据表的创建。

（1）创建数据表“Student”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Student
    -> (
    -> Sno char(8) PRIMARY KEY,
    -> Sname varchar(20) NOT NULL,
    -> Sex ENUM(' 男 ',' 女 '),
    -> Native varchar(30),
    -> Birthday datetime,
    -> Major varchar(20),
    -> Classno char(4),
    -> Tel char(11) UNIQUE,
    -> CONSTRAINT FK_Classno FOREIGN KEY(Classno)
    -> REFERENCES Class(Classno)
    -> );
```

（2）创建数据表“Course”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Course
    -> (
    -> Cno char(5) PRIMARY KEY,
    -> Cname varchar(30) NOT NULL,
    -> Hours tinyint,
    -> Credit tinyint,
    -> Semester tinyint
    -> );
```

（3）创建数据表“Teacher”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Teacher
    -> (
    -> Tno char(8) PRIMARY KEY,
    -> Tname varchar(20) NOT NULL,
    -> Sex ENUM(' 男 ',' 女 '),
    -> Birthday datetime,
    -> Dno varchar(20),
    -> Pno varchar(10),
    -> Tel char(11) UNIQUE,
    -> Email varchar(40) UNIQUE
    -> );
```

（4）创建数据表“Class”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE Class
    -> (
    -> Classno char(4) PRIMARY KEY,
    -> Classname char(16) NOT NULL,
    -> Num int,
    -> Charge varchar(20)
    -> );
```

（5）创建数据表“StuCourse”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE StuCourse
    -> (
    -> Sno char(8) NOT NULL,
    -> Cno char(5) NOT NULL,
    -> Score tinyint,
    -> PRIMARY KEY(Sno,Cno),
    -> CONSTRAINT FK_Sno FOREIGN KEY(Sno)
    -> REFERENCES Student(Sno),
    -> CONSTRAINT FK_Cno FOREIGN KEY(Cno)
    -> REFERENCES Course(Cno)
    -> );
```

笔记

笔记

（6）创建数据表“TeaCourse”，输入的 SQL 语句如下：

```
mysql> CREATE TABLE TeaCourse
    -> (
    -> Tno char(8) NOT NULL,
    -> Classno char(4) NOT NULL,
    -> Cno char(5) NOT NULL,
    -> Semester char(6),
    -> Schoolyear char(10),
    -> PRIMARY KEY(Tno,Classno,Cno),
    -> CONSTRAINT FK_Tno FOREIGN KEY(Tno)
    -> REFERENCES Teacher(Tno),
    -> CONSTRAINT FK_Classno FOREIGN KEY(Classno)
    -> REFERENCES Class(Classno),
    -> CONSTRAINT FK_Cno FOREIGN KEY(Cno)
    -> REFERENCES Course(Cno)
    -> );
```

语句执行完毕后，各数据表创建成功。

拓展阅读

关注数据质量，发挥数据价值

数据质量构成了数据科学与人工智能领域的基石，其重要性随着数据规模的持续扩张而日益凸显。数据质量涵盖数据的准确性、可靠性、一致性、时效性和完整性等多个维度，是衡量数据能否满足特定应用需求的核心指标。数据准确性是指数据对实际情况的正确反映程度；数据可靠性表现在数据不能受到破坏、篡改或非法访问；数据一致性是指数据在不同来源或不同时间点上保持一致；数据时效性则要求数据在特定时间范围内保持有效；数据完整性关注数据是否存在缺失或损坏的情况。

数据质量管理是对数据生命周期每个阶段可能引发的各类数据质量问题进行识别、度量、监控、预警等的一系列管理活动。数据的生命周期通常包含五个阶段：规划设计、创建、使用、老化和消亡，每个环节均可能存在影响数据质量的因素。企业的数据质量管理策略应全面覆盖数据生命周期的每个阶段，从最初的规划设计至最终的数据消亡。通过在各阶段实施细致的管理措施，企业可以预防或减少数据质量问题的发生，确保数据在整个生命周期内保证数据质量，发挥更大的数据价值。

思考与练习

一、简答题

（1）设置外键的作用是什么？

（2）DATETIME 与 TIMESTAMP 都是日期和时间的混合类型，它们的区别是什么？

（3）请说明字符串 CHAR 与 VARCHAR 的区别？

（4）请分别解释 AUTO_INCREMENT、默认值和 NULL 值的用途？

二、拓展题

笔记

在“我为乡村振兴献言”系统数据库“ MessageSys”中，请使用 MySQL 命令创建用户表“User”，表结构如表 4-18 所示，用于保存用户的基本信息；创建留言表“Message”，表结构如表 4-19 所示，用于记录群众留言情况；创建回复留言表“ ReplyMessage”，表结构如表 4-20 所示，用于记录群众回复留言情况。

表 4-18　用户表“User”的结构信息

编号	字段名称	数据类型	要求	说明
1	UserID	char(10)	主键	用户 ID
2	UserName	varchar(50)	非空	用户名
3	Password	varchar(100)	非空	密码
4	Tel	char(11)	唯一	联系电话
5	Email	varchar(50)	唯一	电子邮件
6	Regitime	datetime	—	注册时间
7	Status	char(2)	默认值为“0”	用户状态，“0”代表活跃，“1”代表不活跃

表 4-19　留言表“Message”的结构信息

编号	字段名称	数据类型	要求	说明
1	MessageID	int	主键，自动编号	留言 ID
2	UserID	char(10)	非空，外键	用户 ID
3	Category	varchar(50)	非空	主题类型
4	Subject	varchar(200)	非空	留言标题
5	Words	varchar(1000)	非空	留言内容
6	Attachment	blob	—	图片视频等附件
7	Createtime	datetime	非空	发布时间
8	Updatetime	datetime	—	最近修改时间
9	ReplyNumber	int	默认值为 0	回复数量
10	IsEssence	char(2)	默认值为“N”	是否精品，“Y”代表是，“N”代表否

笔记

表 4-20　回复留言表“ReplyMessage”的结构信息

编号	字段名称	数据类型	要求	说明
1	ReplyID	int	主键，自动编号	回复留言 ID
2	ReplyUserID	char(10)	非空；外键	回复用户 ID
3	MessageID	int	非空；外键	留言 ID
4	ReplyWords	varchar(1000)	非空	回复留言内容
5	Attachment	blob	—	图片、视频等附件
6	Replytime	datetime	非空	回复时间