在线答题

拓展阅读

单元3 数据库和表的基本操作

数据库是数据库管理系统的基础与核心，是存放数据库对象的容器，数据库文件是数据库的存在形式。数据库管理就是设计数据库、定义数据库，以及修改和维护数据库的过程，数据库的效率和性能在很大程度上取决于数据库的设计和优化。本章将详细讲解数据库和数据表的基本操作。

本单元主要学习目标如下：

- 掌握数据库的创建、查看、修改和删除等操作。
- 掌握数据表的创建、查看、修改和删除等操作。
- 了解 MySQL 的数据类型，掌握基本数据类型的使用。
- 掌握表的约束，以及给表添加约束的命令。

3.1 数据库的基本操作

观看视频

3.1.1 创建数据库

MySQL 服务器中的数据库可以有多个，分别存储不同的数据。要想将数据存储到数据库中，首先需要创建数据库，这是使用 MySQL 各种功能的前提。启动并连接 MySQL 服务器，即可对 MySQL 数据库进行操作。

在 MySQL 中创建数据库的基本 SQL 语法格式如下。

```
CREATE DATABASE [IF NOT EXISTS] 数据库名;
```

参数说明如下：

(1) [IF NOT EXISTS]：可选子句，该子句可防止创建数据库服务器中已存在的新数据库的错误，即不能在 MySQL 服务器中创建具有相同名称的数据库。

(2) 数据库名：必选项，即要创建的数据库名称。建议数据库名称尽可能有意义，并且具有一定的描述性。创建数据库时，数据库命名的规则如下。

① 不能与其他数据库重名，否则将发生错误。

② 名称可以由任意字母、阿拉伯数字、下画线和“$”组成，可以使用上述任意字符开头，但不能使用单独的数字开头，否则会造成它与数值相混淆。

③ 名称最长可为 64 个字符。

④ 不能使用 MySQL 关键字作为数据库名。

⑤ 默认情况下，Windows 下对数据库名的大小写不敏感；而在 Linux 下对数据库名的

大小写是敏感的。为了使数据库在不同平台间进行移植，建议采用小写的数据库名。

在创建完数据库后，MySQL 会在存储数据的 data 目录中创建一个与数据库同名的子目录，同时，会在该子目录下生成一个 db.opt 文件，用于保存数据库选项。

【例 3-1】 创建名为 library 的数据库。

(1) 直接使用 CREATE DATABASE 语句创建，SQL 语句如下。

```
CREATE DATABASE library;
```

执行结果如图 3-1 所示。

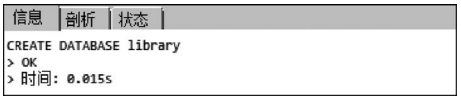


图 3-1　直接使用 CREATE DATABASE 语句创建数据库

(2) 使用含 IF NOT EXISTS 子句的 CREATE DATABASE 语句创建，SQL 语句如下。

```
CREATE DATABASE IF NOT EXISTS library;
```

SQL 语句执行后显示“OK”，说明语句执行成功，数据库已经创建。

3.1.2　查看数据库

在 MySQL 中，成功创建数据库后，可以使用 SHOW DATABASES 语句显示 MySQL 服务器中的所有数据库。语法格式如下。

```
SHOW DATABASES;
```

使用该命令可以查询在 MySQL 中已经存在的所有数据库。

【例 3-2】 在例 3-1 中，我们创建了数据库 library，现在使用命令查看 MySQL 服务器中的所有数据库，SQL 语句如下。

```
SHOW DATABASES;
```

执行结果如图 3-2 所示。

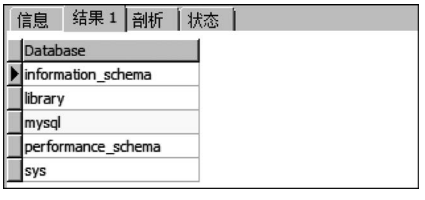


图 3-2　使用 SHOW DATABASES 语句查看数据库

从图 3-2 中可以看出，目前在 MySQL 服务器上存在着 5 个数据库，其中，除例 3-1 中创建的 library 数据库之外，还有 information_schema、mysql、performance_schema、sys 这 4 个数据库，这 4 个数据库都是在 MySQL 安装完成后由系统自动创建的。

(1) information_schema 是信息数据库，存储着 MySQL 数据库服务器所维护的所有其他数据库的信息。在 information_schema 数据库中，有几个只读表。它们实际上是视图，而不是基本表，因此，用户无法看到与之相关的任何文件。

（2）mysql 是 MySQL 的核心数据库，类似于 SQL Server 中的 master 表，主要负责存储数据库的用户、权限设置、关键字等控制和管理信息。mysql 数据库中的数据不可以删除，否则，MySQL 将不能正常运行。如果对 mysql 数据库不是很了解，不要轻易修改这个数据库里的信息。

（3）performance_schema 数据库主要用于收集数据库服务器性能参数。该数据库中所有表的存储引擎均为 performance_schema，而用户是不能创建存储引擎为 performance_schema 的表的。

（4）sys 数据库是一个强大的性能监控和诊断工具，提供视图、函数和过程，用于性能监控、诊断和优化建议。

（5）library 是安装时创建的一个测试数据库，是一个空数据库，其中没有任何表，可以删除。

要想查看某个已经创建的数据库信息，可以通过 SHOW CREATE DATABASE 语句实现，具体语法格式如下。

```
SHOW CREATE DATABASE 数据库名称;
```

【例 3-3】 查看创建好的数据库 library 的信息，SQL 语句如下。

```
SHOW CREATE DATABASE library;
```

执行之后，输出结果显示了数据库 library 的创建信息及其编码方式。

3.1.3 选择数据库

上面虽然成功创建了数据库 library，但并不表示当前就可以使用数据库 library。在使用指定数据库之前，必须使用 USE 语句告诉 MySQL 要使用哪个数据库，使其成为当前默认数据库。其语法格式如下。

```
USE 数据库名;
```

【例 3-4】 选择名称为 library 的数据库，设置其为当前默认的数据库。使用 USE 语句选择数据库 library，SQL 语句如下。

```
USE library;
```

执行结果如图 3-3 所示。

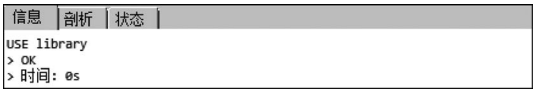


图 3-3 选择名称为 library 的数据库


观看视频

3.1.4 修改数据库

数据库创建之后，数据库编码方式就确定了。修改数据库的编码方式，可以使用 ALTER DATABASE 语句，具体语法格式如下。

```
ALTER DATABASE 数据库名称 DEFAULT CHARACTER SET 编码方式 COLLATE 编码方式_bin;
```

其中“数据库名称”是要修改的数据库的名称，“编码方式”是修改后的数据库编码方式。

【例 3-5】　将数据库 library 的编码方式修改为 gbk，SQL 语句如下。

```
ALTER DATABASE library DEFAULT CHARACTER SET gbk COLLATE gbk_bin;
```

为了验证数据库的编码方式是否修改成功，可以使用例 3-3 中的 SHOW CREATE DATABASE 语句查看修改后的数据库。

3.1.5　删除数据库

删除数据库可以使用 DROP DATABASE 命令，具体语法格式如下。

```
DROP DATABASE 数据库名称;
```

其中“数据库名称”是要删除的数据库的名称。需要注意的是，如果要删除的数据库不存在，则会出现错误。

【例 3-6】　删除名为 company 的数据库，SQL 语句如下。

```
DROP DATABASE company;
```

执行结果如图 3-4 所示。

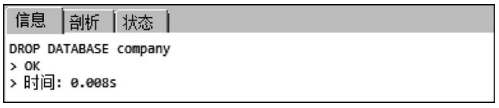


图 3-4　DROP DATABASE 语句的执行结果

为了验证数据库是否删除成功，可以使用 SHOW DATABASES 语句查看当前 MySQL 数据库服务器上的所有数据库，当前 MySQL 数据库服务器中已经不存在 company 数据库，表明删除成功。

注意：在使用 DROP DATABASE 删除数据库时，若待删除的数据库不存在，则 MySQL 服务器会报错。值得一提的是，在执行删除数据库操作之前，一定要备份需要保留的数据，确保数据的安全，避免误操作造成严重后果。

注意：MySQL 中单行注释以“#”开始标识，也支持标准 SQL 中“--”单行注释。但是为了防止“--”与 SQL 语句中负号和减法运算的混淆，在第二个短横线后必须添加至少一个控制字符（如空格、制表符、换行符等）将其标识为单行注释符号。示例如下。

```
#此处填写单行注释内容，如：若服务器中没有 mydb 数据库，则创建，否则忽略此 SQL
CREATE DATABASE IF NOT EXISTS mydb;
-- 此处填写单行注释内容，如：若服务器中存在 mydb 数据库，则删除，否则忽略此 SQL
DROP DATABASE IF EXISTS mydb;
```

同样地，MySQL 也支持标准 SQL 中的多行注释“/* 此处填写注释内容 */”，它的开始符号为“/*”，结束符号为“*/”，中间的内容就是要编写的注释。示例如下。

```
/*
此处填写多行注释内容
如：利用以下 SQL 查看当前服务器中的所有数据库
*/
SHOW DATABASES;
```

在开发中编写的 SQL 语句，建议合理地添加单行或多行注释，方便阅读与理解。

在 MySQL 使用的过程中，相关的基本语法有以下 3 点需要注意。

（1）换行、缩进与结尾分隔符。MySQL 中的 SQL 语句可以单行或多行书写，多行书写时可以按 Enter 键换行，每行中的 SQL 语句可以使用空格和缩进增强语句的可读性，在 SQL 语句完成时通常情况下使用分号（;）结尾，在命令行窗口中也可使用“\g”结尾，效果与分号相同。另外，在命令行窗口中，还可以使用“\G”结尾，如 SHOW DATABASES\G 将显示结果以每条记录（一行数据）为一组，将所有的字段纵向排列展示。

（2）大小写问题。MySQL 的关键字在使用时不区分大小写，如 SHOW DATABASES 与 show databases 都表示获取当前 MySQL 服务器中有哪些数据库。另外，MySQL 中的所有数据库名称、数据表名称、字段名称默认情况下在 Windows 系统下都忽略大小写，在 Linux 系统下数据库与数据表名称则区分大小写，通常开发时推荐都使用小写。

（3）反引号的使用。在项目开发中，为了避免用户自定义的名称与系统中的命令（如关键字）冲突，最好使用反引号（``）包裹数据库名称、字段名称和数据表名称。其中，反引号（`）在键盘中左上角 Tab 键的上方，读者只需将输入法切换到英文，按下此键即可输入反引号（`）。


观看视频

3.2 数据类型

在 MySQL 中，为字段选择合适的数据类型对数据库的优化非常重要。MySQL 支持多种数据类型，大致可以分为 4 类：数值类型、日期和时间类型、字符串（字符）类型和二进制类型。

1. 数值类型

MySQL 支持所有标准 SQL 数值类型，包括精确数值类型（INTEGER、SMALLINT 和 DECIMAL）和近似数值类型（FLOAT、REAL 和 DOUBLE PRECISION）。关键字 INT 是 INTEGER 的简写，关键字 DEC 是 DECIMAL 的简写。

作为 SQL 标准的扩展，MySQL 也支持整数类型 TINYINT、MEDIUMINT 和 BIGINT。表 3-1 列出了 MySQL 每种数值类型占用的字节数、范围以及用途。

表 3-1 数值类型

类　型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1b	(−128,127)	(0,255)	小整数值
SMALLINT	2b	(−32 768,32 767)	(0,65 535)	大整数值
MEDIUMINT	3b	(−8 388 608,8 388 607)	(0,16 777 215)	大整数值
INT 或 INTEGER	4b	(−2 147 483 648,2 147 483 647)	(0,4 294 967 295)	大整数值
BIGINT	8b	(−9 223 372 036 854 775 808, 9 223 372 036 854 775 807)	(0,18 446 744 073 709 551 615)	极大整数值
FLOAT	4b	(−3.402 823 466 E+38, 1.175 494 351 E−38)	(1.175 494 351 E−38, 3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8b	(−1.797 693 134 862 315 7 E+308, −2.225 073 858 507 201 4 E−308)	(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	M+2	(−1.797 693 134 862 315 7 E+308, −2.225 073 858 507 201 4 E−308)	(2.225 073 858 507 201 4 E−308, 1.797 693 134 862 315 7 E+308)	小数值

从表 3-1 中可以看出，不同数值类型所占用的字节数和取值范围都是不同的。其中，定

点数类型 DECIMAL 的有效取值范围由 M 和 D 决定，M 表示整个数据的位数，不包括小数点；D 表示小数点后数据的位数。例如，将数据类型为 DECIMAL(5,3)的数据 3.1415 插入数据库，显示的结果为 3.142。

创建表时，选择数值类型应遵循以下原则。

(1) 选择最小的可用类型，如果值永远不超过 127，则使用 TINYINT 比使用 INT 强。

(2) 对于完全都是数字的，可以选择整型数据。

(3) 浮点型数据用于可能具有小数部分的数，如货物单价、网上购物支付金额等。

注意：

(1) 在选择数据类型时，若一个数据将来可能参与数学计算，推荐使用整数、浮点数或定点数类型；若只用来显示，则推荐使用字符串类型。例如，商品库存可能需要增加、减少、求和等，所以保存为整数类型；用户的身份证、电话号码一般不需要计算，可以保存为字符串类型。

(2) 表的主键推荐使用整数类型，与字符串类型相比，整数类型的处理效率更高，查询速度更快。

(3) 当插入的值的数据类型与字段的数据类型不一致，或使用 ALTER TABLE 修改字段的数据类型时，MySQL 会尝试尽可能将现有的值转换为新类型。例如，字符串'123'、'－123'、'1.23'与数字 123、－123、1.23 可以互相转换；1.5 转换为整数时，会被四舍五入，结果为 2。

2. 日期和时间类型

表示日期和时间值的日期和时间类型有 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。每个时间类型有一个有效值范围和一个“零”值，当输入不合法的值时，MySQL 使用“零”值插入。TIMESTAMP 类型具备专有的自动更新特性。表 3-2 列举了 MySQL 中日期和时间类型所对应的字节数、范围、格式以及用途。

表 3-2　日期和时间类型

类　型	大小	范　围	格　式	用　途
DATE	3b	1000-01-01～9999-12-31	YYYY-MM-DD	日期值
TIME	3b	'－838:59:59'～'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1b	1901～2155	YYYY	年份值
DATETIME	8b	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	4b	1970-01-01 00:00:00～2038 结束时间是第 2 147 483 647 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	YYYY-MM-DD HH:MM:SS	混合日期和时间值，时间戳

其中，DATE 类型用于表示日期值，不包含时间部分。在 MySQL 中，DATE 类型常用的字符串格式为"YYYY-MM-DD"或者"YYYYMMDD"。

例如，输入'2023-08-24'或者'20230824'，插入数据库的日期都为'2023-08-24'。

TIME 类型用于表示时间值，它的显示形式一般为 HH:MM:SS，其中，HH 表示小时，MM 表示分，SS 表示秒。

例如，输入'115253'，插入数据库中的时间为 11:52:53。

YEAR 类型用于表示年。在 MySQL 中，常使用 4 位字符串或数字表示，对应的字符串的范围为'1901'～'2155'，数字范围为 1901～2155。

例如，输入'2023'或 2023，插入数据库中的值均为 2023。

DATETIME 类型用于表示日期和时间，它的显示形式为“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，SS 表示秒。

例如，输入'2023-08-24 08:23:52'或者'20230824082352'，插入数据库中的 DATETIME 类型的值均为 2023-08-24 08:23:52。

TIMESTAMP 类型用于表示日期和时间，它的显示形式与 DATETIME 类型相同，但取值范围比 DATETIME 类型小。当 TIMESTAMP 类型的字段输入为 NULL 时，系统会以当前系统的日期和时间填入。当 TIMESTAMP 类型的字段无输入时，系统也会以当前系统的日期和时间填入。

3. 字符串类型和二进制类型

为了存储字符串、图片和声音等数据，MySQL 提供了字符串类型和二进制类型。表 3-3 列举了这些数据类型的取值范围和用途。

表 3-3 字符串类型和二进制类型

类 型	取值范围	用 途
CHAR(*n*)	0～255 字符	定长字符串，*n* 为字符串的最大长度。若输入数据的长度超过了 *n* 值，超出部分将会被截断；否则，不足部分用空格填充。例如，对于 CHAR(4)，若插入值为'abc'，则其占用的存储空间为 4 字节
VARCHAR(*n*)	0～65 536 字符	变长字符串，*n* 为字符串的最大长度。占用字节数随输入数据的实际长度而变化，最大长度不得超过 *n*+1。例如，对于 VARCHAR(4)，若插入值为'abc'，则其占用的存储空间为 4 字节，若插入值为'abcd'，则其占用的存储空间为 5 字节
BINARY(*n*)	0～255 字节	固定长度的二进制数据，*n* 为字节长度，若输入数据的字节长度超过了 *n* 值，超出部分将会被截断；否则，不足部分用字符'\0'填充。例如，对于 BINARY(3)，插入值为'a\0'时，会变成'a\0\0'值存入
VARBINARY(*n*)	0～65 536 字节	可变长度的二进制数据，*n* 为字节长度
ENUM	1～65 535 个值	枚举类型，语法格式为：ENUM('值 1','值 2',…,'值 *n*')。该类型的字段值只能为枚举值中的某一个。例如，性别字段数据类型可以设为 ENUM('男','女')
SET	1～64 个值	集合类型，语法格式为：SET('值 1', '值 2',…,'值 *n*')。例如，人的兴趣爱好字段的数据类型可以设为 SET('听音乐','看电影','购物','游泳','旅游')，该字段的值从集合中取值，且可以取多个值
BIT(*n*)	1～64 位	位字段类型。如果输入的值的长度小于 *n* 位，在值的左边用 0 填充。例如，为 BIT(6)分配值'101'的效果与分配值'000101 的效果相同
TINYBLOB	0～255 字节	不超过 255 个字符的二进制字符串

续表

类　型	取值范围	用　途
BLOB	0～65 535 字节	二进制形式的文本数据，主要存储图片、音频等信息
MEDIUMBLOB	0～16 777 215 字节	二进制形式的中等长度文本数据
LONGBLOB	0～4 294 967 295 字节	二进制形式的极大长度文本数据
TINYTEXT	0～255 字节	短文本字符串
TEXT	0～65 535 字节	文本数据。如新闻内容、博客、日志等
MEDIUMTEXT	0～16 777 215 字节	中等长度文本数据
LONGTEXT	0～4 294 967 295 字节	极大长度文本数据

CHAR(n)为固定长度的字符串，在定义时指定字符串的长度最大为 n 个字符个数。当保存时，MySQL 会自动在右侧填充空格，以达到其指定的长度。例如，CHAR(8)定义了一个固定长度的字符串列，字符个数最大为 8。当检索到 CHAR 值时，尾部的空格将被删除。

VARCHAR(n)为可变长度的字符串。VARCHAR 的最大实际长度 L 由最长行的大小和使用的字符集确定，其实际占用的存储空间为字符串实际长度 L 加 1。例如，VARCHAR(50)定义了一个最大长度为 50 的字符串列，如果写入的实际字符串只有 20 个字符，则其实际存储的字符串为 20 个字符和一个字符串结束字符。保存和检索 VARCHAR 的值时，其尾部的空格仍然保留。

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和尾部空格是否被保留等也不同。在存储或检索过程中不进行大小写转换。TEXT 类型用于保存非二进制字符串，如文章的内容、评论等信息。当保存或检索 TEXT 列的值时，不会删除尾部空格。

TEXT 类型有 4 种：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。不同的 TEXT 类型的存储空间和数据长度都不同。

MySQL 中的二进制字符串数据类型有 BIT、BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。

BINARY 和 VARBINARY 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。

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。它们的区别是可容纳的存储范围不同。

创建表时，使用字符串类型时应遵循以下原则。

(1) 从速度方面考虑，要选择固定的列，可以使用 CHAR 类型。

(2) 要节省空间，使用动态的列，可以使用 VARCHAR 类型。

(3) 要将列中的内容限制为一种选择，可以使用 ENUM 类型。

(4) 允许在一列中有多个条目，可以使用 SET 类型。

(5) 如果要搜索的内容不区分大小写，可以使用 TEXT 类型。

(6) 如果要搜索的内容区分大小写，可以使用 BLOB 类型。

3.3 数据表的基本操作

3.3.1 创建数据表


观看视频

创建完数据库并熟悉了 MySQL 支持的数据类型后，接下来的工作是创建数据表。创建数据表其实就是在已经创建好的数据库中建立新表。

数据表属于数据库，在创建数据表之前，应该使用语句“USE <数据库名>”指定操作是在哪个数据库中进行。如果没有选择数据库，MySQL 会抛出 No database selected 的错误提示。

创建数据表的语句为 CREATE TABLE，语法规则如下。

```
CREATE TABLE 数据表名称
(
字段名 1 数据类型 [完整性约束条件] [默认值],
字段名 2 数据类型 [完整性约束条件] [默认值],
…
字段名 n 数据类型 [完整性约束条件] [默认值]
);
```

在上述语法格式中，“数据表名称”是创建的数据表的名称，“字段名”是数据表的列名，“完整性约束条件”是字段的特殊约束条件。关于表的约束将在 3.4 节进行详细讲解。

使用 CREATE TABLE 创建表时，必须注意以下几点。

(1) 数据表名不区分大小写，且不能使用 SQL 中的关键字，如 DROP、INSERT 等。

(2) 如果数据表中有多个字段(列)，字段(列)的名称和数据类型要用英文逗号隔开。

【例 3-7】 在 library 数据库中创建一个用于存储图书信息的 books 表，其结构如表 3-4 所示。

表 3-4 books 表结构

字 段 名	数 据 类 型	备 注 说 明
Bookid	char(6)	图书编号
Bookname	varchar(50)	书名
Author	varchar(50)	作者
Press	varchar(40)	出版社
Pubdate	date	出版日期
Type	varchar(20)	类型
Number	int(2)	库存数量
Info	varchar(255)	简介

使用 library 数据库，SQL 语句如下。

```
USE library;
```

接下来创建 books 表，SQL 语句如下。

```
CREATE TABLE books
(
    Bookid char(6),
```

```
    Bookname varchar(50),
    Author varchar(50),
    Press varchar(40),
    Pubdate date,
    Type varchar(20),
    Number int(2),
    Info varchar(255)
);
```

执行语句后，便创建了一个名称为 books 的数据表，使用 SHOW TABLES 语句查看数据表是否创建成功，SQL 语句如下。

```
SHOW TABLES;
```

执行结果如图 3-5 所示。

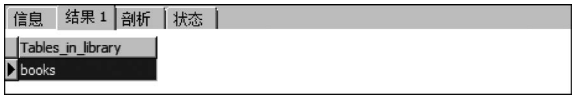


图 3-5　使用 SHOW TABLES 语句查看数据表

从图 3-5 中可以看到，library 数据库中已经有了数据表 books，表明数据表创建成功。

3.3.2　查看数据表

创建好数据表之后，可以查看数据表，以确认其定义是否正确。在 MySQL 中，查看数据表的方式有以下两种。

1. 使用 SHOW CREATE TABLE 语句查看数据表

语法格式如下。

```
SHOW CREATE TABLE 数据表名称;
```

其中，“数据表名称”是要查看的数据表的名称。

【例 3-8】 使用 SHOW CREATE TABLE 语句查看 books 表，SQL 语句如下。

```
SHOW CREATE TABLE books;
```

执行结果如图 3-6 所示。

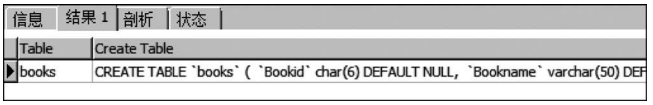


图 3-6　使用 SHOW CREATE TABLE 语句查看 books 表的详细结构

2. 使用 DESCRIBE 语句查看数据表

使用 DESCRIBE 语句查看数据表，可以查看到数据表的字段名、类型、是否为空、是否为主键等信息。语法格式如下。

```
DESCRIBE 表名;
```

或者使用简写，语法格式如下。

```
DESC 表名;
```

【例 3-9】 使用 DESCRIBE 语句查看 books 表，SQL 语句如下。

```
DESCRIBE books;
```

执行结果如图 3-7 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(50)	YES		(Null)	
Author	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-7 使用 DESCRIBE 语句查看 books 表的基本结构

其中：

(1) Field 表示该表的字段名。

(2) Type 表示对应字段的数据类型。

(3) Null 表示对应字段是否可以存储 NULL 值。

(4) Key 表示对应字段是否编制索引和约束。

(5) Default 表示对应字段是否有默认值。

(6) Extra 表示获取到的与对应字段相关的附加信息。


观看视频

3.3.3 修改数据表

数据表创建之后，用户还可以对表中的某些信息进行修改，修改表指修改数据库中已经存在的数据表结构。MySQL 使用 ALTER TABLE 语句修改表。常用的修改表的操作有：修改表名、修改字段名和数据类型、修改字段的数据类型、添加字段、删除字段、修改字段的排列位置等。下面对这些操作进行详细介绍。

1. 修改表名

MySQL 可以通过 ALTER TABLE 语句实现对表名的修改，语法格式如下。

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

其中，关键字 TO 是可选的，使用与否都不影响运行结果。

【例 3-10】 将数据库 library 中 books 表的表名改为 tb_books。

修改表名之前，先用 SHOW TABLES 语句查看数据库中的表，执行结果如图 3-8 所示。

信息 | 结果 1 | 剖析 | 状态

Tables_in_library
books

图 3-8 使用 SHOW TABLES 语句查看数据库中表的执行结果

执行下述命令，将 books 表名改为 tb_books。

```
ALTER TABLE books RENAME tb_books;
```

上述命令执行成功后，再用 SHOW TABLES 语句查看数据库中的表，执行结果如图 3-9 所示。

图 3-9　改名后使用 SHOW TABLES 语句查看数据库中表的执行结果

从图 3-9 中可以看出，数据库 library 中的表名 books 已经被成功修改为 tb_books。

2. 修改字段名和数据类型

语法格式如下。

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

其中，“旧字段名”是修改之前的字段名称，“新字段名”是修改之后的字段名称，“新数据类型”是修改后的数据类型。注意，修改后的数据类型不能为空。如果只修改字段名，不修改数据类型，可以将新数据类型写为字段原来的数据类型。

【例 3-11】 将 tb_books 表中的 Author 字段改名为 bookAuthor，数据类型保持不变。修改字段之前，用 DESC tb_books 查看表的信息，执行结果如图 3-10 所示。

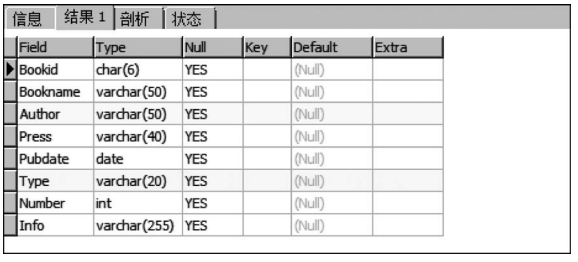


图 3-10　执行修改字段名前查看表结构

执行下述命令，将 tb_books 表中的 Author 字段改名为 bookAuthor。

```
ALTER TABLE tb_books CHANGE Author bookAuthor varchar(50);
```

执行结果如图 3-11 所示。

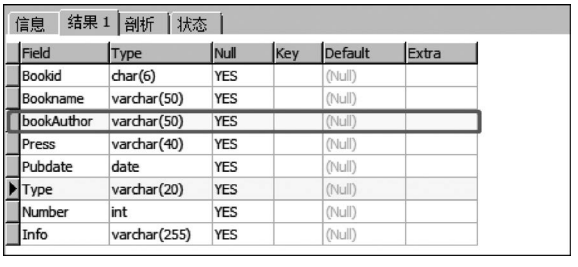


图 3-11　执行修改字段名后查看表结构

从图 3-11 中可以看出，tb_books 表中的 Author 字段成功改名为 bookAuthor。

3. 修改字段的数据类型

语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型;
```

【例 3-12】 将 tb_books 表中的 Bookname 字段的数据类型由 varchar(50)修改为 varchar(100)。执行修改命令之前，先用 DESC tb_books 语句查看 tb_books 表的结构，如图 3-12 所示。

执行修改命令，SQL 语句如下。

```
ALTER TABLE tb_books MODIFY Bookname varchar(100);
```

命令成功执行后，再查看一下 tb_books 表的结构，执行结果如图 3-13 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(50)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-12 执行修改字段数据类型前查看表结构

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(100)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-13 执行修改字段数据类型后查看表结构

从图 3-13 中可以看出，Bookname 字段的数据类型已经由 varchar(50)修改为 varchar(100)。

4. 添加字段

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 已经存在的字段名];
```

其中，“新字段名”是新添加的字段名称；“FIRST”是可选参数，用于将新添加的字段设置为表的第一个字段；“AFTER 已经存在的字段名”也是可选参数，用于将新添加的字段添加到指定字段的后面。如不指定位置，则默认将新添加字段追加到表末尾。

【例 3-13】 在数据表 tb_books 中的 Press 字段后添加一个 int 类型的字段 column1，SQL 语句如下。

```
ALTER TABLE tb_books ADD column1 int AFTER Press;
```

为了验证 column1 字段是否添加成功，使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-14 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(100)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
column1	int	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-14 执行添加字段后查看表结构的执行结果

从图 3-14 中可以看出，在 tb_books 表中已经成功添加了 column1 字段，数据类型为 int。

5. 删除字段

语法格式如下。

```
ALTER TABLE 表名 DROP 字段名;
```

【例 3-14】 删除 tb_books 表中的 column1 字段，SQL 语句如下。

```
ALTER TABLE tb_books DROP column1;
```

为了验证 column1 字段是否删除成功，使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-15 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(100)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-15　执行删除字段后查看表结构的执行结果

从图 3-15 中可以看出，tb_books 表中的 column1 字段已经不存在了。

6. 修改字段的排列位置

在创建一个数据表时，字段的排列位置就已经确定了，但这个位置并不是不能改变的，可以使用 ALTER TABLE 改变指定字段的位置，语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 1 新数据类型 FIRST|AFTER 字段名 2;
```

其中，“FIRST”是可选参数，用于将“字段名 1”设置为表的第一个字段；“AFTER 字段名 2”也是可选参数，用于将“字段名 1”移动到“字段名 2”的后面。此命令可以同时修改字段的数据类型和位置。如果只修改位置，不修改数据类型，可以将新数据类型写为字段原来的数据类型。

【例 3-15】 将 tb_books 表中的 Press 字段修改为表中的第一个字段。

SQL 语句如下。

```
ALTER TABLE tb_books MODIFY Press varchar(40) FIRST;
```

使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-16 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Press	varchar(40)	YES		(Null)	
Bookid	char(6)	YES		(Null)	
Bookname	varchar(100)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-16　执行修改 Press 字段为第一个字段后查看表结果

从图 3-16 中可以看出，Press 字段已经被修改为表的第一个字段了。

【例 3-16】 将 tb_books 表中的 Press 字段移动到 bookAuthor 字段之后。

SQL 语句如下。

```
ALTER TABLE tb_books MODIFY Press Tinyint AFTER bookAuthor;
```

使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-17 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(100)	YES		(Null)	
bookAuthor	varchar(50)	YES		(Null)	
press	tinyint	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-17 执行移动 Press 字段后查看表结果

从图 3-17 中可以看出，tb_books 表中的 Press 字段已经成功移动到 bookAuthor 字段之后，并且数据类型已被修改为 tinyint。

3.3.4 删除数据表

删除数据表指删除数据库中已经存在的表，同时，该数据表中的数据也会被删除。注意，一般数据库中的多个数据表之间可能会存在关联，要删除具有关联关系的数据表比较复杂，这种情况将在后续章节介绍。本节只涉及没有关联关系的数据表的删除。

删除数据表的语法格式如下。

```
DROP TABLE 表名;
```

【例 3-17】 删除 tb_books 表。

SQL 语句如下。

```
DROP TABLE tb_books;
```

为了验证 tb_books 表是否删除成功，使用 DESC 语句查看，执行结果如图 3-18 所示。

信息 | 状态

```
DESC tb_books
> 1146 - Table 'library.tb_books' doesn't exist
> 时间: 0.001s
```

图 3-18 使用 DESC 语句查看 DROP TABLE 语句的执行结果

从图 3-18 中可以看出，tb_books 表已经不存在了，说明已被成功删除。

3.4 数据表的约束

MySQL 对数据库表中数据的插入、更新和删除操作强加了规则或条件，即表的约束，用于保证数据的完整性和一致性。常见的表约束有以下 5 种：主键约束（PRIMARY KEY CONSTRAINT）、外键约束（FOREIGN KEY CONSTRAINT）、非空约束（NOT NULL

CONSTRAINT)、唯一约束(UNIQUE CONSTRAINT)和默认约束(DEFAULT CONSTRAINT)。

3.4.1　主键约束

主键又称主码,由表中的一个字段或多个字段组成,能够唯一地标识表中的一条记录。主键约束要求主键字段中的数据唯一,并且不允许为空。主键分为两种类型:单字段主键和复合主键。

注意,每个数据表中最多只能有一个主键。


观看视频

1. 单字段主键

(1) 创建表时指定主键,语法格式如下。

```
字段名　数据类型　PRIMARY KEY;
```

【例 3-18】 创建 tb_books 表,并设置 Bookid 字段为主键,SQL 语句如下。

```
CREATE TABLE tb_books
(
    Bookid char(6) PRIMARY KEY,
    Bookname varchar(50),
    Author varchar(50),
    Press varchar(40),
    Pubdate date,
    Type varchar(20),
    Number int(2),
    Info varchar(255)
);
```

执行上述命令之后,用 DESC 语句查看 tb_books 表的结构,执行结果如图 3-19 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	NO	PRI	(Null)	
Bookname	varchar(50)	YES		(Null)	
Author	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-19　设置 Bookid 为主键后的表结构

从图 3-19 中可以看出,Bookid 字段的"Key"显示为 PRI,表示此字段为主键。

(2) 为已存在的表添加主键约束,语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 数据类型 PRIMARY KEY;
```

【例 3-19】 将 tb_books 表的 Bookid 字段修改为主键。

首先将前面创建的 tb_books 表删除,再新建 tb_books 表,SQL 语句如下。

```
DROP TABLE tb_books;
CREATE TABLE tb_books
(
    Bookid char(6) ,
```

```
    Bookname varchar(50),
    Author varchar(50),
    Press varchar(40),
    Pubdate date,
    Type varchar(20),
    Number int(2),
    Info varchar(255)
);
```

执行上述命令后，用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-20 所示。

接下来，使用 ALTER 语句将 Bookid 字段修改为主键，SQL 语句如下。

```
ALTER TABLE tb_books MODIFY Bookid char(6) PRIMARY KEY;
```

为了验证 Bookid 字段的主键约束是否添加成功，再次使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-21 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	YES		(Null)	
Bookname	varchar(50)	YES		(Null)	
Author	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-20　删除并创建表 tb_books 后的执行结果

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	NO	PRI	(Null)	
Bookname	varchar(50)	YES		(Null)	
Author	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-21　使用 ALTER 语句修改表主键的执行结果

从图 3-21 中可以看出，Bookid 字段的“Key”显示为 PRI，表示此字段为主键。

（3）删除主键约束，语法格式如下。

```
ALTER TABLE 表名 DROP PRIMARY KEY;
```

【例 3-20】 删除 tb_books 表的 Bookid 字段的主键约束，SQL 语句如下。

```
ALTER TABLE tb_books DROP PRIMARY KEY;
```

为了验证 Bookid 字段的主键约束是否删除，使用 DESC 语句查看 tb_books 表的结构，执行结果如图 3-22 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
Bookid	char(6)	NO		(Null)	
Bookname	varchar(50)	YES		(Null)	
Author	varchar(50)	YES		(Null)	
Press	varchar(40)	YES		(Null)	
Pubdate	date	YES		(Null)	
Type	varchar(20)	YES		(Null)	
Number	int	YES		(Null)	
Info	varchar(255)	YES		(Null)	

图 3-22　删除主键约束的执行结果

从图 3-22 中可以看出，Bookid 字段的“Key”为空，表示此字段已经不是主键了。

2. 复合主键

复合主键指主键由多个字段组成。

（1）创建表时指定复合主键，其语法格式如下。

```
PRIMARY KEY (字段名 1, 字段名 2,…,字段名 n);
```

其中，“字段名 1，字段名 2，…，字段名 n”指的是构成主键的多个字段的名称。

【例 3-21】 创建 sales 表，设置 product_id、region_code 字段为复合主键，SQL 语句如下。

```
CREATE TABLE sales
(
    product_id INT(11),
    region_code varchar(10),
    quantity int(11),
    price float,
    PRIMARY KEY (product_id,region_code)
);
```

执行上述命令后，用 DESC 语句查看 sales 表的结构，执行结果如图 3-23 所示。

从图 3-23 中可以看出，product_id 字段和 region_code 字段的“Key”均显示为 PRI，表示这两个字段共同作为主键。

（2）为已存在的表添加复合主键，语法格式如下。

```
ALTER TABLE 表名 ADD PRIMARY KEY (字段名 1, 字段名 2,…,字段名 n);
```

【例 3-22】 将 sales 表的 product_id 字段和 region_code 字段作为复合主键。

首先将前面创建的 sales 表删除，再新建 sales 表，SQL 语句如下。

```
DROP TABLE sales;
CREATE TABLE sales
(
    product_id int(11),
    region_code varchar(10),
    quantity int(11),
    price float,
);
```

执行上述命令后，用 DESC 语句查看 sales 表的结构，执行结果如图 3-24 所示。

信息　结果 1　剖析　状态

Field	Type	Null	Key	Default	Extra
product_id	int	NO	PRI	(Null)	
region_code	varchar(10)	NO	PRI	(Null)	
quantity	int	YES		(Null)	
price	float	YES		(Null)	

图 3-23　指定复合主键的执行结果

信息　结果 1　剖析　状态

Field	Type	Null	Key	Default	Extra
product_id	int	YES		(Null)	
region_code	varchar(10)	YES		(Null)	
quantity	int	YES		(Null)	
price	float	YES		(Null)	

图 3-24　删除并新建 sales 表的执行结果

接下来，使用 ALTER 语句将 sales 表的 product_id 字段和 region_code 字段设为复合主键，SQL 语句如下。

```
ALTER TABLE sales ADD PRIMARY KEY (product_id, region_code);
```

为了验证 product_id 字段和 region_code 字段作为复合主键是否添加成功，再次使用 DESC 语句查看 sales 表的结构，执行结果如图 3-25 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
product_id	int	NO	PRI	(Null)	
region_code	varchar(10)	NO	PRI	(Null)	
quantity	int	YES		(Null)	
price	float	YES		(Null)	

图 3-25　使用 ALTER 语句修改 sales 表主键的执行结果

（3）删除复合主键约束，语法格式如下。

```
ALTER  TABLE 表名  DROP  PRIMARY  KEY;
```


观看视频

3.4.2　外键约束

外键用来在两个表的数据之间建立关联，它可以是一个字段或者多个字段。一个表可以有一个或者多个外键。外键对应的是参照完整性，一个表的外键可以为空值，若不为空值，则每一个外键值必须等于另一个表中主键的某个值。注意，关联指的是在关系数据库中，相关表之间的联系。它是通过相同或者相容的字段或字段组来表示的。从表的外键必须关联主表的主键，且关联字段的数据类型必须匹配。

定义外键后，不允许在主表中删除与子表具有关联关系的记录。

主表（父表）：对于两个具有关联关系的表而言，相关联字段中主键所在的那个表即主表。

从表（子表）：对于两个具有关联关系的表而言，相关联字段中外键所在的那个表即从表。

1. 创建表时添加外键约束

语法格式如下。

```
CONSTRAINT 外键名 FOREIGN KEY (从表的外键字段名) REFERENCES 主表名 (主表的主键字段名)
```

其中，“外键名”指从表创建的外键约束的名字。

【例 3-23】 定义借书表 borrow，其表结构见表 3-5；在 borrow 表上创建外键约束，其中图书表 books 为主表，borrow 表为从表。

表 3-5　borrow 表结构

序号	列　名	数据类型	备　注
1	Borrowid	char(6)	借书记录号
2	Borrowbookid	char(6)	图书编号
3	Borrowreaderid	char(6)	借书证号
4	Borrowdate	date	借阅时间
5	Borrownum	Int(2)	借阅册数

定义数据表 borrow，让它的键 Borrowbookid 作为外键关联到 books 表的主键 Bookid，SQL 语句如下。

```
CREATE TABLE borrow
(
    Borrowid char(6) PRIMARY KEY,
    Borrowbookid char(6),
```

```
    Borrowreaderid char(6),
    Borrowdate datetime,
    Borrownum int(2),
    CONSTRAINT fk_bks_brw FOREIGN KEY(Borrowbookid) REFERENCES books(Bookid)
)ENGINE = InnoDB;
```

执行上述命令后，使用 SHOW CREATE TABLE 语句查看 books 表的结构，执行结果如图 3-26 所示。

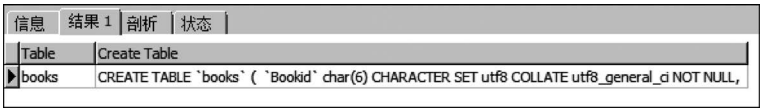


图 3-26　使用 SHOW CREATE TABLE 语句查看 books 表的执行结果

使用 SHOW CREATE TABLE 语句查看 borrow 表的结构，执行结果如图 3-27 所示。

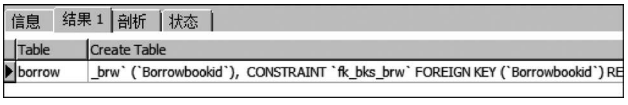


图 3-27　使用 SHOW CREATE TABLE 语句查看 borrow 表的执行结果

从图 3-27 中可以看出已经成功地创建了 books 表和 borrow 表的主外键关联。要特别注意，主表 books 的主键字段 Bookid 和从表 borrow 的外键字段 Borrowbookid 的数据类型必须兼容或者一致，且含义一样。在创建表时创建表的主外键关联，必须先创建主表，再创建从表。

2. 为已存在的表添加外键约束

语法格式如下。

```
ALTER TABLE 从表名 ADD CONSTRAINT 外键名 FOREIGN KEY (从表的外键字段名) REFERENCES 主表名(主表的主键字段名);
```

其中，“外键名”指从表创建的外键约束的名字。

【例 3-24】 已存在图书表 books 和借阅表 borrow，为借阅表 borrow 创建外键。

(1) 删除 borrow 表和 books 表，SQL 语句如下。

```
DROP TABLE borrow;
DROP TABLE books;
```

(2) 先创建主表 books，SQL 语句如下。

```
CREATE TABLE books
(
    Bookid char(6) PRIMARY KEY,
    Bookname varchar(50),
    Author varchar(50),
    Press varchar(40),
    Pubdate date,
    Type varchar(20),
    Number int(2),
    Info varchar(255)
)ENGINE = InnoDB;
```

（3）再创建从表 borrow，SQL 语句如下。

```
CREATE TABLE borrow
(
    Borrowid char(6) PRIMARY KEY,
    Borrowbookid char(6),
    Borrowreaderid char(6),
    Borrowdate datetime,
    Borrownum int(2)
)ENGINE = InnoDB;
```

执行上述命令后，使用 SHOW CREATE TABLE 语句查看 books 表，执行结果如图 3-28 所示。

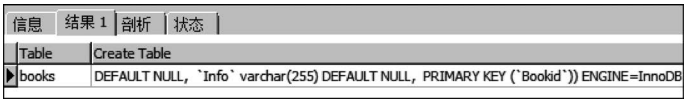


图 3-28　使用 SHOW CREATE TABLE 语句查看 books 表的执行结果

使用 SHOW CREATE TABLE 语句查看 borrow 表，执行结果如图 3-29 所示。

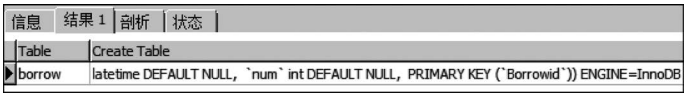


图 3-29　使用 SHOW CREATE TABLE 语句查看 borrow 表的执行结果

接下来，使用 ALTER 语句为 borrow 表创建外键，SQL 语句如下。

```
ALTER TABLE borrow ADD CONSTRAINT fk_bks_brw FOREIGN KEY(Borrowbookid) REFERENCES books
(Bookid);
```

为了验证借阅表 borrow 的外键是否创建成功，再次使用 SHOW CREATE TABLE 语句查看 borrow 表，执行结果如图 3-30 所示。

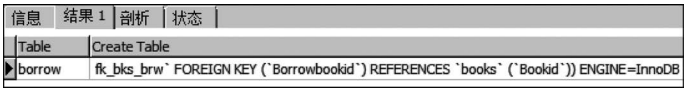


图 3-30　使用 SHOW CREATE TABLE 语句查看添加外键的执行结果

对比图 3-29 和图 3-30，可以看出已经成功地创建了 books 表和 borrow 表的主外键关联。

3. 删除外键约束

语法格式如下。

```
ALTER TABLE 从表名 DROP FOREIGN KEY 外键名;
```

【例 3-25】 删除 borrow 表 Borrowbookid 字段的外键约束，外键约束名是 fk_bks_brw，SQL 语句如下。

```
ALTER TABLE borrow DROP FOREIGN KEY fk_bks_brw;
```

为了验证 borrow 表 Borrowbookid 字段的外键约束是否删除，使用 SHOW CREATE TABLE 语句查看 borrow 表，执行结果如图 3-31 所示。

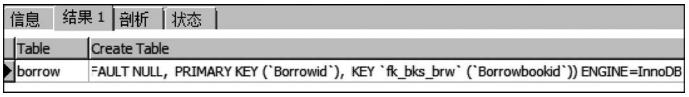

信息 | 结果 1 | 剖析 | 状态

Table	Create Table
borrow	=AULT NULL, PRIMARY KEY (`Borrowid`), KEY `fk_bks_brw` (`Borrowbookid`)) ENGINE=InnoDB

图 3-31　使用 SHOW CREATE TABLE 语句查看删除外键执行结果

对比图 3-30 和图 3-31，可以看出已经成功地删除了 books 表和 borrow 表的主外键关联。但是仍出现“Key 'fk_bks_brw'('Borrowbookid')”的信息，是因为 MySQL 在创建外键后，会自动创建一个同名的索引。外键删除，但索引不会被删除。本书会在后续章节中详细介绍索引。

3.4.3　非空约束


观看视频

非空约束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，数据库系统会报错。

1. 创建表时添加非空约束

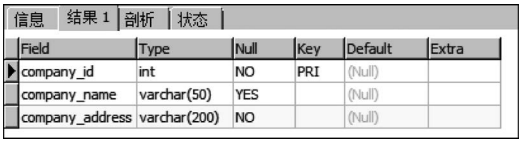
语法格式如下。

```
字段名 数据类型 NOT NULL;
```

【例 3-26】 创建 company 表，并设置 company_id 字段为主键，company_address 字段为非空约束，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50),
    company_address varchar(200) NOT NULL
);
```

执行上述命令后，使用 DESC 语句查看 company 表的结构，执行结果如图 3-32 所示。


信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-32　创建 company 表时添加非空约束的执行结果

从图 3-32 中可以看出，company_address 字段的“Null”列的值为 NO，表示这个字段不允许为空。

2. 为已经存在的表添加非空约束

语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型 NOT NULL;
```

此命令可以同时修改字段的数据类型和增加非空约束。如果不修改字段的数据类型，将“新数据类型”写为字段原来的数据类型即可。

【例 3-27】 将 company 表的 company_address 字段设置为非空约束。

首先创建 company 表，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50),
    company_address varchar(200)
);
```

执行上述命令后，使用 DESC 语句查看 company 表的结构，执行结果如图 3-33 所示。

接下来，使用 ALTER 语句将 company_address 字段设置为非空约束，SQL 语句如下。

```
ALTER TABLE company MODIFY company_address VARCHAR(200) NOT NULL;
```

为了验证 company_address 字段的非空约束是否添加成功，再次使用 DESC 语句查看 company 表的结构，执行结果如图 3-34 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	YES		(Null)	

图 3-33　创建 company 表的执行结果

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-34　company 表添加非空约束的执行结果

从图 3-34 中可以看出，company_address 字段的“Null”列的值为 NO，表示这个字段不允许为空。

3. 删除非空约束

语法格式如下。

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

【例 3-28】 删除 company 表的 company_address 字段的非空约束。

SQL 语句如下。

```
ALTER TABLE company MODIFY company_address varchar(200);
```

为了验证 company_address 字段的非空约束是否删除成功，使用 DESC 语句查看 company 表的结构，执行结果如图 3-35 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	YES		(Null)	

图 3-35　company 表删除非空约束的执行结果

从图 3-35 中可以看出，company_address 字段的“Null”列的值为 YES，表示这个字段允许为空。

观看视频

3.4.4　唯一约束

唯一约束要求该列值唯一，不能重复。唯一约束可以确保一列或者几列不出现重复值。

1. 创建表时添加唯一约束

语法格式如下。

```
字段名 数据类型 UNIQUE;
```

【例 3-29】 创建 company 表，并将 company_id 字段设置为主键，company_address 字段设置为非空约束，company_name 字段设置为唯一约束。

SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50) UNIQUE,
    company_address varchar(200) NOT NULL
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-36 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-36　创建表时添加唯一约束的执行结果

从图 3-36 中可以看出，company_name 字段的“Key”列的值为 UNI，表示这个字段具有唯一约束。

注意：一张表中可以有多个字段声明为唯一约束，但是只能有一个主键；声明为主键的字段不允许有空值，但是声明为唯一约束的字段允许存在空值。

2. 为已存在的表添加唯一约束

语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型 UNIQUE;
```

此命令可以同时修改字段的数据类型和增加唯一约束。如果不修改字段的数据类型，将“新数据类型”写为字段原来的数据类型即可。

【例 3-30】 将 company 表的 company_name 字段修改为唯一约束。首先创建 company 表，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50),
    company_address varchar(200) NOT NULL
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-37 所示。

接下来，使用 ALTER 语句为 company_name 字段添加唯一约束，SQL 语句如下。

```
ALTER TABLE company MODIFY company_name varchar(50) UNIQUE;
```

为了验证 company_name 字段的唯一约束是否添加成功，再次使用 DESC 语句查看 company 表的结构，执行结果如图 3-38 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-37　创建表 company 的执行结果

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-38　为 company 表添加唯一约束的执行结果

从图 3-38 中可以看出，company_name 字段的“Key”列的值为 UNI，表示这个字段具有唯一约束。

3. 删除唯一约束

语法格式如下。

```
ALTER TABLE 表名 DROP INDEX 字段名;
```

【例 3-31】 删除 company 表的 company_name 字段的唯一约束，SQL 语句如下。

```
ALTER TABLE company DROP INDEX company_name;
```

为了验证 company_name 字段的唯一约束是否删除成功，使用 DESC 语句查看 company 表的结构，执行结果如图 3-39 所示。从图 3-39 中可以看出，company_name 字段的“Key”列的值为空，表示这个字段已没有唯一约束。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES		(Null)	
company_address	varchar(200)	NO		(Null)	

图 3-39　在 company 表中删除唯一约束的执行结果


观看视频

3.4.5　默认约束

若将数据表中某列定义为默认约束，在用户插入新的数据行时，如果没有为该列指定数据，那么数据库系统会自动将默认值赋给该列，默认值也可以是空值(NULL)。

1. 创建表时添加默认约束

语法格式如下。

```
字段名 数据类型 DEFAULT 默认值;
```

【例 3-32】 创建 company 表，并将 company_id 字段设置为主键，company_address 字段设置为非空约束，company_name 字段设置为唯一约束，company_tel 字段的默认值为“0371-”，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50) UNIQUE,
    company_address varchar(200) NOT NULL,
```

```
    company_tel varchar(20) DEFAULT '0371-'
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-40 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-40　例 3-32 的执行结果

从图 3-40 中可以看出，company_tel 字段的“Default”列的值为“0371-”，表示这个字段具有默认值“0371-”。

2. 为已存在的表添加默认约束

语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型 DEFAULT 默认值;
```

此命令可以同时修改字段的数据类型和增加默认约束。如果不修改字段的数据类型，将“新数据类型”写为字段原来的数据类型即可。

【例 3-33】 为 company 表的 company_tel 字段添加默认约束，默认值为“0371-”。

首先创建 company 表，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50) UNIQUE,
    company_address varchar(200) NOT NULL,
    company_tel varchar(20)
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-41 所示。

接下来，使用 ALTER 语句为 company_tel 字段添加默认约束，SQL 语句如下。

```
ALTER TABLE company MODIFY company_tel varchar(20) DEFAULT '0371-';
```

为了验证 company_tel 字段的默认约束是否添加成功，再次使用 DESC 语句查看 company 表的结构，执行结果如图 3-42 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		(Null)	

图 3-41　创建 company 表的执行结果

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-42　company 表添加默认约束的执行结果

从图 3-42 中可以看出，company_tel 字段的“Default”列的值为“0371-”，表示这个字段具有默认值“0371-”。

3. 删除默认约束

语法格式如下。

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

【例 3-34】 删除 company 表的 company_tel 字段的默认约束，SQL 语句如下。

```
ALTER TABLE company MODIFY company_tel varchar(20);
```

为了验证 company_tel 字段的默认约束是否删除，使用 DESC 语句查看 company 表的结构，执行结果如图 3-43 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		(Null)	

图 3-43 删除 company 表默认约束的执行结果

从图 3-43 中可以看出，company_tel 字段已经没有默认值了。


观看视频

3.5 字段值自动增加

在数据库中，如果表的主键值是逐一增加的，希望在每次插入记录时由系统自动生成，则可以通过为表的主键添加 AUTO_INCREMENT 关键字来实现。在 MySQL 中，AUTO_INCREMENT 字段的初始值是 1，每增加一条记录，字段值自动加 1，但一张表只能有一个字段使用 AUTO_INCREMENT 约束，且该字段必须设置为主键。AUTO_INCREMENT 约束所在的字段可以是任何整数类型（TINYINT，SMALLINT，INT，BIGINT）。

1. 创建表时指定字段值自动增加

语法格式如下。

```
字段名 数据类型 PRIMARY KEY AUTO_INCREMENT;
```

【例 3-35】 创建 company 表，并将 company_id 字段设置为主键，其值自动增加，company_address 字段设置为非空约束，company_name 字段设置为唯一约束，company_tel 字段的默认值设置为“0371-”，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY AUTO_INCREMENT,
    company_name varchar(50) UNIQUE,
    company_address varchar(200) NOT NULL,
    company_tel varchar(20) DEFAULT '0371 - '
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-44 所示。

从图 3-44 中可以看出，company_id 字段的“Extra”列的值为“auto_increment”，表示这个字段值是自动增加的。系统会自动填入自动增加字段的值，用户在插入记录时不需要给出。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	auto_increment
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-44　创建表时指定字段值自动增加的执行结果

2. 为已存在的表设置字段值自动增加

语法格式如下。

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型 AUTO_INCREMENT;
```

【例 3-36】 设置 company 表的 company_id 字段值自动增加。

首先创建 company 表，SQL 语句如下。

```
DROP TABLE IF EXISTS company;
CREATE TABLE company
(
    company_id int(11) PRIMARY KEY,
    company_name varchar(50) UNIQUE,
    company_address varchar(200) NOT NULL,
    company_tel varchar(20) DEFAULT '0371 - '
);
```

执行上述命令后，用 DESC 语句查看 company 表的结构，执行结果如图 3-45 所示。

接下来，使用 ALTER 语句将 company_id 字段设置为自动增加，SQL 语句如下。

```
ALTER TABLE company MODIFY company_id int(11) AUTO_INCREMENT;
```

为了验证 company_id 字段值的自动增加是否添加成功，再次使用 DESC 语句查看 company 表的结构，执行结果如图 3-46 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-45　创建 company 表的执行结果

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	auto_increment
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-46　company 表设置自动增加的执行结果

从图 3-46 中可以看出，company_id 字段的“Extra”列的值为“auto_increment”，表示这个字段值是自动增加的。

3. 删除字段值的自动增加

语法格式如下。

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

【例 3-37】 删除 company 表的 company_id 字段值的自动增加，SQL 语句如下。

```
ALTER TABLE company MODIFY company_id int(11);
```

为了验证 company_id 字段值的自动增加是否删除，使用 DESC 语句查看 company 表的结构，执行结果如图 3-47 所示。

信息 | 结果 1 | 剖析 | 状态

Field	Type	Null	Key	Default	Extra
company_id	int	NO	PRI	(Null)	
company_name	varchar(50)	YES	UNI	(Null)	
company_address	varchar(200)	NO		(Null)	
company_tel	varchar(20)	YES		0371-	

图 3-47　删除字段值的自动增加的执行结果

从图 3-47 中可以看出，company_id 字段的“Extra”列的值为空，表示这个字段值不再自动增加。

注意：在为字段删除自动增加并重新添加自动增长后，自动增长的初始值会自动设为该列现有的最大值加 1。在修改自动增加值时，修改的值若小于该列现有的最大值，则修改不会生效。

观看视频

3.6　综合案例：教务管理系统数据库

本节以教务管理系统为例来介绍数据库的创建和数据表的设计。学生可以通过教务管理系统查看所有选修课程的相关信息，包括课程号、课程名、学时、学分，然后选择要选修的课程；学生也可以通过系统查看相关授课教师的信息，包括工号、姓名、性别、学历、职称。教师可以通过系统查看选修自己课程的学生的信息，包括学号、姓名、性别、出生日期、班级。在考试结束后，教师可以通过系统录入学生的考试成绩，学生可以通过系统查看自己的考试成绩。

3.6.1　创建“教务管理系统”数据库

“教务管理系统”数据库的创建语句如下。

```
CREATE DATABASE JwSystem;
```

3.6.2　在“教务管理系统”数据库中创建表

根据教务管理系统的要求，在 jwsystem 数据库中设计如下数据表。

（1）学生表的结构设计如表 3-6 所示。

表 3-6　学生表（studentInfo）结构

序号	列名	数据类型	允许 NULL 值	约束	备注
1	sno	char(8)	不能为空	主键	学号
2	sname	varchar(10)	不能为空		姓名
3	sgender	char(1)			性别
4	sbirth	date			出生日期
5	sclass	varchar(20)			班级

创建 studentInfo 表的 SQL 语句如下。

```
CREATE TABLE studentInfo
(
    sno char(8) PRIMARY KEY NOT NULL,
```

```
    sname varchar(10) NOT NULL,
    sgender char(2),
    sbirth DATE,
    sclass varchar(20)
);
```

(2) 教师表的结构设计如表 3-7 所示。

表 3-7　教师表(teacher)结构

序号	列　　名	数据类型	允许 NULL 值	约　　束	备注
1	tno	char(4)	不能为空	主键	工号
2	tname	varchar(10)	不能为空		姓名
3	tgender	char(1)			性别
4	tedu	varchar(10)			学历
5	tpro	varchar(8)		默认为“副教授”	职称

创建 teacher 表的 SQL 语句如下。

```
CREATE TABLE teacher
(
    tno char(4) PRIMARY KEY NOT NULL,
    tname varchar(10) NOT NULL,
    tgender char(2),
    tedu varchar(10),
    tpro varchar(8) DEFAULT '副教授'
);
```

(3) 课程表的结构设计如表 3-8 所示。

表 3-8　课程表(course)结构

序号	列　　名	数据类型	允许 NULL 值	约　　束	备注
1	cno	char(4)	不能为空	主键	课程号
2	cname	varchar(40)		唯一约束	课程名
3	cperiod	int			学时
4	credit	decimal(3,1)			学分
5	ctno	char(4)		是教师表的外键	授课教师

创建 course 表的 SQL 语句如下。

```
CREATE TABLE course
(
    cno char(4) PRIMARY KEY NOT NULL,
    cname varchar(40) UNIQUE,
    cperiod int,
    credit decimal(3,1),
    ctno char(4) ,
    CONSTRAINT fk_teacher_course FOREIGN KEY (ctno) REFERENCES teacher(tno)
);
```

（4）成绩表的结构设计如表 3-9 所示。

表 3-9　成绩表(score)结构

序号	列名	数据类型	允许 NULL 值	约　束	备注
1	sno	char(8)		主键(学号,课程号),其中学号是学生表的外键,课程号是课程表的外键	课程号
2	cno	char(4)			课程名
3	score	int			成绩

创建 score 表的 SQL 语句如下。

```
CREATE TABLE score
(
    sno char(8),
    cno char(4),
    score int,
    PRIMARY KEY (sno,cno),
    CONSTRAINT fk_course_score FOREIGN KEY (cno) REFERENCES course (cno),
    CONSTRAINT fk_stu_score FOREIGN KEY (sno) REFERENCES studentInfo (sno)
);
```

“教务管理系统”数据库创建完毕。

单元小结

本单元介绍了创建数据库、查看数据库、修改数据库、删除数据库等数据库的基本操作；讲解了创建数据表、查看数据表、修改数据表、删除数据表等数据表的基本操作；并介绍了数据库的数据类型和 MySQL 中的 5 种约束：主键约束、外键约束、非空约束、唯一约束、默认约束。

单元实训项目

项目一：创建“网上书店”数据库

在安装好的 MySQL 中创建“网上书店”数据库(如：BookShop)。

项目二：在“网上书店”数据库中创建表

目的：

(1) 熟练掌握创建表结构的方法。

(2) 掌握查看表信息的方法。

内容：

(1) 使用 MySQL 创建会员表(见表 3-10)、图书表(见表 3-11)的表结构。

表 3-10　会员表(user)结构

列　名	数据类型	允许 NULL 值	约　束	备　注
uid	char(4)	不允许	主键	会员编号
uname	varchar(20)			会员昵称
email	varchar(20)			电子邮箱
tnum	varchar(15)			联系电话
score	int			积分

表 3-11　图书表(book)结构

列　名	数据类型	允许 NULL 值	约　束	备　注
bid	int	不允许	主键	图书编号
bname	varchar(50)	不允许		图书名称
author	char(8)			作者
price	float			价格
publisher	varchar(50)			出版社
discount	float			折扣
cid	int		图书类别表的外键	图书类别

(2) 使用 MySQL 创建图书类别表(见表 3-12)、订购表(见表 3-13)的表结构。

表 3-12　图书类别表(category)结构

列　名	数据类型	允许 NULL 值	约　束	备　注
cid	int	不允许	主键	类别编号
cname	varchar(16)			类别名称

表 3-13　订购表(b_order)结构

列　名	数据类型	允许 NULL 值	约　束	备　注
bid	int	不允许		图书编号
uid	char(4)	不允许		会员编号
ordernum	int		默认值为 1	订购量
orderdate	datetime			订购日期
deliverydate	datetime			发货日期

(3) 使用 DROP TABLE 语句删除上述创建的表,然后使用 CREATE TABLE 语句再次创建上述表。

(4) 查看会员表的信息。

(5) 修改会员表结构。添加字段“联系地址”,数据类型设置为 varchar(50);更改“联系地址”为“联系方式”;删除添加的字段“联系地址”。

(6) 使用创建表时添加约束和为已存在的表添加约束这两种方式给表添加约束。

单元练习题

一、选择题

1. 下列(　　)不能用作 MySQL 数据库名。

A. minrisoft　　B. mingrisoft_01　　C. com$com　　D. 20170609

2. 下列(　　)语句不是创建数据库的语句。

A. CREATE SCHEMA　　B. CREATE DATABASE

C. CREATE TABLE　　D. ALTER TABLE

3. 下列(　　)语句可以用于查看服务器中所有的数据库名称。

A. SHOW DATABASE　　B. SHOW DATABASES

C. SHOW ENGINES　　D. SHOW VARIABLES

4. 下列(　　)语句可以用于将 db_library 数据库作为当前默认的数据库。

A. CREATE DATABASE db_library　　B. SHOW db_library

C. USE db_library　　D. SELECT db_library

5. 下列关于数据类型的选择方法描述错误的是(　　)。

A. 选择最小的可用类型,如果值永远不超过 127,则使用 TINYINT 比 INT 强

B. 对于完全都是数字的,可以选择整数类型

C. 浮点类型用于可能具有小数部分的数

D. 以上都不对

6. UNIQUE 唯一索引的作用是(　　)。

A. 保证各行在该索引上的值不能为 NULL

B. 保证各行在该索引上的值都不能重复

C. 保证唯一索引不能被删除

D. 保证参加唯一索引的各列,不能再参加其他的索引

7. 创建数据表时,使用(　　)语句。

A. ALTER TABLE　　B. CREATE DATABASE

C. CREATE TABLE　　D. ALERT DATABASE

8. 下列(　　)不是 MySQL 常用的数据类型。

A. INT　　B. VARCHAR　　C. CHAR　　D. MONEY

9. 想要删除数据库中已经存在的数据表,可以使用(　　)语句。

A. CREATE TABLE　　B. DROP DATABASE

C. ALERT TABLE　　D. DROP TABLE

10. 在 MySQL 中,非空约束可以通过(　　)关键字定义。

A. NOT NULL　　B. DEFAULT

C. CHECK　　D. UNIQUE

二、判断题

1. MySQL 数据库一旦安装成功,创建的数据库编码也就确定了,是不可以更改的。

(　　)

2. 在 MySQL 中，如果添加的日期类型不合法，系统将报错。（　　）
3. 在删除数据表的时，如果表与表之间存在关系，那么可能导致删除失败。（　　）
4. 一张数据表中可以有多个主键约束。（　　）

三、简答题

1. 简述主键的作用及其特征。
2. 创建、查看、修改、删除数据库的语句是什么？
3. 创建、查看、修改、删除数据表的语句是什么？
4. 数据表有哪些约束？写出为数据表添加约束的语句。
5. 什么是非空约束？写出其基本语法格式。
6. 什么是默认约束？写出其基本语法格式。