第5章

MySQL数据表管理

CHAPTER 5

“数接千载，据联万里。”数据表由表名、表中的字段和表的记录3部分组成。设计数据表结构就是定义数据表文件名、确定数据表包含哪些字段和各字段的字段名、选择合适的数据类型及宽度，并将这些数据输入计算机中。本章将通过丰富的案例和7个综合课业任务分别演示数据库中数据表的创建、使用、修改、删除操作以及通过不同的工具管理MySQL数据表。

【教学目标】

- 熟悉常见的MySQL数据类型、约束类型；
- 掌握如何创建数据表、查看数据表的结构、修改数据表；
- 掌握通过不同的工具管理MySQL数据表。

【课业任务】

王小明想利用MySQL+Java开发一个数据库学习系统，在熟悉了MySQL数据库的管理后，需熟练掌握对数据表的管理，为后续开发数据库学习系统打下良好的基础。现通过7个课业任务来完成。

*课业任务5-1　创建用户登录表

课业任务5-2　向用户登录表中添加字段

课业任务5-3　修改用户登录表中字段的数据类型

课业任务5-4　删除用户登录表的一个字段

课业任务5-5　删除用户登录表

课业任务5-6　使用MySQL Workbench工具创建用户登录表

课业任务5-7　使用Navicat Premium工具向用户登录表添加字段

5.1 数据类型

5.1.1 MySQL 数据类型介绍

数据类型用于在系统中限制或允许该列中存储的数据。在 MySQL 中,数据类型主要根据数据值的内容、大小、精度来选择,为字段选择合适的数据类型对数据库的优化具有重要作用;反之,则可能会严重影响应用程序的功能和性能。MySQL 支持多种数据类型,主要分为 3 种:数值、日期与时间、字符串类型。其中,数值类型包括整数、浮点数和定点数类型;字符串类型包括文本字符串、二进制字符串类型。MySQL 数据类型如表 5-1 所示。

表 5-1 MySQL 数据类型

类型名称	数据类型
整数类型	TINYINT、SMALLINT、MEDIUMINT、INT(INTEGER)、BIGINT
浮点数类型	FLOAT、DOUBLE
定点数类型	DECIMAL
日期与时间类型	YEAR、TIME、DATE、DATETIME、TIMESTAMP
文本字符串类型	CHAR、VARCHAR、TEXT、MEDIUMTEXT、LONGTEXT、ENUM、SET
二进制字符串类型	BINARY、VARBINARY、BLOB、BIT

5.1.2 整数类型

MySQL 中的整数类型分为 TINYINT、SMALLINT、MEDIUMINT、INT(INTEGER)和 BIGINT 这 5 个类型。不同的数据类型存储空间不同,提供的取值范围也不同。因为存储范围越大,存储的空间也越大,所以在实际中根据需求选择合适的数据类型,这有利于节约存储空间以及利于提高查询效率。MySQL 整数类型如表 5-2 所示。

表 5-2 MySQL 整数类型

整数类型	说明	字节数	有符号数值取值范围	无符号数值取值范围
TINYINT	非常小的整数	1	−128～127	0～255
SMALLINT	小整数	2	−32768～32767	0～65535
MEDIUMINT	中等大小的整数	3	−8388608～8388607	0～16777215
INT(INTEGER)	普通大小的整数	4	−2147483648～2147473647	0～4294967295
BIGINT	非常大的整数	8	−9223372036854775808～9223372036854775807	0～18446744073709551615

说明:在 MySQL 5.7 版本中,可以在定义表结构时指定整数数据类型所需要的显示宽度,如果不指定,则系统为每种类型指定默认的宽度值。从 MySQL 8.0.17 开始,整数数据类型不推荐使用显示宽度属性。

面对实际场景需求时,该如何进行选择?MySQL 整数类型的不同场景选择如下。

(1) TINYINT:一般用于枚举数据,如系统设定取值范围很小且固定的场景。

(2) SMALLINT:一般用于较小范围的统计数据,如统计工厂的固定资产库存数量等。

(3) MEDIUMINT:一般用于较大整数的计算,如车站每日的客流量等。

(4) INT(INTEGER):取值范围足够大,一般情况下不用考虑超限问题,用得最多,如商品编号。

(5) BIGINT：一般只有当处理特别巨大的整数时才会用到，如“双十一”的电商交易量、大型门户网站点击量、证券公司衍生产品持仓等。

5.1.3　浮点数类型与定点数类型

1. 浮点数类型

在实际开发中，很多情况下需要存储的数据是有小数数值的，就要使用到浮点数类型。MySQL 中的浮点数类型主要有两种，分别单精度浮点数 FLOAT 和双精度浮点数 DOUBLE。

浮点数类型可以用(M,D)来表示，其中 M 称为精度，表示整数的位数；D 称为标度，表示小数的位数。MySQL 浮点数类型如表 5-3 所示。

表 5-3　MySQL 浮点数类型

浮点数类型	说明	字节数	有符号数值取值范围	无符号数值取值范围
FLOAT(M,D)	单精度浮点数	4	−3.402823466E+38 ～ −1.1754943511E−38	0 和 1.1754943511E−38～ 3.402823466E+38
DOUBLE(M,D)	双精度浮点数	8	−1.7976931348623157E+308 ～ −2.225073858507201 4E−308	0 和−2.2250738585072014E−308 ～1.7976931348623157E+308

说明：

(1) FLOAT 和 DOUBLE 浮点数类型的区别：FLOAT 占用字节数少，取值范围小；DOUBLE 占用字节数多，取值范围也大。

(2) 当浮点数类型不指定数据精度时，系统会默认按照实际计算机硬件和操作系统决定精度；若指定精度超出浮点数类型的数据精度，系统则会自动四舍五入，且正常显示。

2. 定点数类型

当项目对精确度要求较高时，则可以使用定点数类型。MySQL 中只有 DECIMAL 一种定点数类型，定点数也可以用(M,D)来表示，其中 M 称为精度，表示数据的总位数；D 称为标度，表示数据的小数部分的位数。MySQL 定点数类型如表 5-4 所示。

表 5-4　MySQL 定点数类型

定点数类型	字　节　数	无符号数值取值范围
DECIMAL(M,D),DEC	M+2	有效范围内由 M 和 D 决定

说明：

(1) 定点数类型是以字符串存储的。

(2) 当定点数类型不指定 M 和 D 时，系统则默认为 DECIMAL(10,0)。

(3) 若数据的精度超出了定点数类型的精度范围，系统也会进行四舍五入操作，但会有警告。

在实际场景当中，该如何进行选择浮点数和定点数类型？MySQL 浮点数和定点数类型的不同场景选择如下。

(1) 浮点数类型适用于取值范围大，且可容忍微小误差的科学计算场景，如计算化学、分子建模、流体动力学等。

(2) 定点数类型适用于对精度要求极高的场景，如涉及金额计算的场景。

5.1.4　日期与时间类型

MySQL 有多种数据类型用于表示日期和时间，主要有 YEAR、TIME、DATE、

DATETIME 和 TIMESTAMP 类型。MySQL 日期与时间类型如表 5-5 所示。

表 5-5　MySQL 日期与时间类型

日期与时间类型	说明	字节数	日 期 格 式	数值取值范围
YEAR	年	1	YYYY 或 YY	1901～2155
TIME	时间	3	HH:MM:SS	－838:59:59～838:59:59
DATE	日期	3	YYYY-MM-DD	1000-01-01～9999-12-03
DATETIME	日期时间	8	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00～9999-12-31 23:59:59
TIMESTAMP	日期时间	4	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 UTC～2038-01-19 03:14:07UTC

1. YEAR 类型

YEAR 类型有两种存储格式,分别是以 4 位字符串或数字格式表示和以两位字符串格式表示。

说明:

(1) 当以 4 位字符串或数字格式表示时,格式为 YYYY,取值范围为 1901～2155。

(2) 当以两位字符串格式表示 YEAR 类型时,表示范围如表 5-6 所示。

表 5-6　YEAR 类型(两位字符串格式)

YY 取值	表 示 范 围
01～69	2001～2069
70～99	1970～1999
日期/字符串"0"(整数 0 或 00)	2000(0000)

2. DATE 类型

DATE 类型用于表示仅需要日期信息的值,没有时间部分,格式为 YYYY-MM-DD,其中 YYYY 表示年,MM 表示月,DD 表示日。

说明:

(1) 若以 YYYYMMDD 格式表示,则会被转换为 YYYY-MM-DD 格式。

(2) 使用 CURRENT_DATE()或 NOW()函数,会获取当前系统的日期。

3. TIME 类型

TIME 类型用于表示只需要时间信息的值,没有日期部分,格式为 HH: MM: SS,其中 HH 表示小时,MM 表示分钟,SS 表示秒。

说明:

(1) 如果使用带有 D 的字符串,如 D HH:MM:SS、D HH:MM 等格式,当插入字段时,D(表示天)会被转换为小时,计算方法为 D×24＋HH。

(2) 当使用带有冒号并且不带 D 的字符串表示时间时,如 12:34:56,表示当天的时间; 不带有冒号的字符串或数字,如"123456"或 123456,格式为"HHMMSS"或 HHMMSS,将被自动转换为 HH:MM:SS 格式进行存储。如果插入一个不合法的字符串或数字,如 12:34:56 PM 或 1234567,则会将其自动转换为 00:00:00 进行存储。因为在 MySQL 中,时间类型的数据是用 HH:MM:SS 格式进行存储和比较的,如果插入的数据不符合这个格式,MySQL 会将其自动转换为 HH:MM:SS 格式,如果无法转换,则会被视为 00:00:00。

(3) 使用 CURRENT_TIME()或 NOW()函数,会插入当前系统的日期。

4. DATETIME 类型

DATETIME 类型在格式上是 DATE 类型和 TIME 类型的结合，是在所有类型中存储空间最大的，格式为 YYYY-MM-DD HH:MM:SS 或 YYYYMMDDHHMMSS，其中 YYYY 表示年，前 1 个 MM 表示月，DD 表示日，HH 表示小时，后 1 个 MM 表示分钟，SS 表示秒。

说明：

(1) 插入 DATETIME 类型的字段时，两位数的年份规则符合 YEAR 类型的规则。

(2) 与 DATE 类型的存储格式类似，以 YYYYMMDDHHMMSS 格式插入 DATETIME 类型的字段时，会被转换为 YYYY-MM-DD HH:MM:SS 格式。

(3) 使用 CURRENT_TIMESTAMP()或 NOW()函数，可以向 DATETIME 类型的字段插入当前系统的日期和时间。

5. TIMESTAMP 类型

TIMESTAMP 类型与 DATETIME 类型的格式相同，也可以表示日期和时间。但与 DATETIME 类型不同的是，TIMESTAMP 类型是以 UTC(世界标准时间)格式进行存储的，存储时对当前时区进行转换，查询时再转换回当前时区，也就是在不同地区查询时会显示不同时间。

说明：

(1) 当插入 TIMESTAMP 类型的字段时，两位数值的年份同样符合 YEAR 类型的规则条件。

(2) TIMESTAMP 类型表示的时间范围要小很多，在插入字段时，不要超出范围，否则 MySQL 会抛出错误。

(3) 使用 CURRENT_TIMESTAMP()或 NOW()函数，可以向 TIMESTAMP 类型的字段插入当前系统的日期和时间。

在实际场景中，该如何选择日期与时间的数据类型？MySQL 日期与时间类型在实际场景中的选择如下。

(1) 若存储数据需要记录年份，则使用 YEAR 类型。

(2) 若存储数据只需要记录时间，则使用 TIME 类型。

(3) 若需要同时记录日期和时间，则可以使用 TIMESTAMP 或 DATETIME 类型。

(4) DATETIME 类型占 8 字节，TIMESTAMP 类型占 4 字节，若要求存储范围较大，建议使用 DATETIME 类型。DATETIME 类型反映的是插入时当地的时区，不会因为访问用户时区不同显示的结果发生变化；而 TIMESTAMP 类型反映的是访问用户的时区，不同时区的用户访问会显示不同的结果。使用 DATETIME 和 TIMESTAMP 类型比较大小或计算日期时，TIMESTAMP 类型会更快、更方便。

5.1.5　文本字符串类型

MySQL 支持的字符串类型包括文本字符串类型和二进制字符串类型，主要用来存储字符串数据，以及存储图片和声音的二进制数据。MySQL 中的文本字符串类型主要包括 CHAR、VARCHAR、TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT、ENUM、SET 等。

MySQL 字符串数据类型如表 5-7 所示，其中 M 表示为其指定的长度。

表 5-7 MySQL 字符串类型

数据类型	说明	长度范围	字节数
CHAR(M)	固定长度非二进制字符串	0≤M≤255	M
VARCHAR(M)	可变长非二进制字符串	0≤M≤65535	M+1
TINYTEXT	小文本,可变长度	0≤L≤255	L+2
TEXT	文本,可变长度	0≤L≤65535	L+2
MEDIUMTEXT	中等文本,可变长度	0≤L≤16777215	L+3
LONGTEXT	大文本,可变长度	0≤L≤4294967295(相当于 4GB)	L+4
ENUM	枚举类型,只能有一个枚举字符串类型	0≤L≤65535	1 或 2
SET	一个设置,字符串对象可以有 0 个或多个 SET 成员	0≤L≤64	1,2,3,4 或 8

每种文本字符串类型的长度范围和占用存储空间都是不同的,在实际应用中要考虑好该字段适合的长度和存储空间,再选择合适的数据类型。

1. CHAR 类型与 VARCHAR 类型

在 MySQL 中,CHAR(M)类型一般需要先定义字符串长度 M,若没有指定 M,则表示长度默认是 1 个字符;而 VARCHAR(M)类型在定义时必须指定长度 M,否则会报错。

说明:

(1) 当检索到 CHAR 类型的数据时,CHAR 类型字段尾部的空格将被删除。

(2) VARCHAR 类型在保存和检索字段数据时,字段尾部的空格仍会保留。

在实际场景中,该如何进行选择 CHAR 和 VARCHAR 类型? MySQL 中 CHAR 和 VARCHAR 类型在实际场景中的选择如下。

(1) 当存储的信息较短,速度要求高时,可以使用 CHAR 类型实现,如班级号(01,02,…);反之,则选择 VARCHAR 类型实现。

(2) 当需要固定长度时,使用 CHAR 类型会更合适,而 VARCHAR 类型可变长的特性就消失,而且还会占多一个长度信息。由于 CHAR 类型平均占用的空间大于 VARCHAR 类型,所以除了简短并且固定长度的情况,其他考虑使用 VARCHAR 类型。

(3) 在 InnoDB 存储引擎中,建议使用 VARCHAR 类型。因为对于 InnoDB 数据表,内部的行存储格式并没有区分固定长度和可变长度列,而且主要影响性能的因素是数据行使用的存储总量,由于 VARCHAR 类型是按实际长度进行存储的,这样节省空间,磁盘 I/O 和数据存储总量性能比较好。

2. TEXT 类型

在 MySQL 中,TEXT 类型分为 4 种,分别为 TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT 类型,不同的 TEXT 类型保存的数据长度和所占用的存储空间都不同。当在 TEXT 类型字段中保存或查询数据时,与 VARCHAR 类型相同,不会删除数据尾部的空格。

在实际场景中,该如何进行选择 TEXT 类型? MySQL 中 TEXT 类型在实际场景中的选择如下。

(1) 当数据列保存非二进制字符串时,如文章内容、评论等。

(2) 在实际开发当中,实际存储长度不确定时,不建议使用 TEXT 类型字段作主键。

(3) 当字符数大于 5000 时,建议使用 TEXT 类型,并且新建一个表进行存储,避免影响索

引查询效率。

3. ENUM 类型

ENUM 类型又叫作枚举类型，它的取值范围需要在创建表时通过枚举方式进行指定，在设置字段值时，ENUM 类型只允许从成员中选取单个值，不能一次选取多个值，其所需要的存储空间由定义 ENUM 类型时指定的成员个数决定。ENUM 类型如表 5-8 所示，其中 L 表示实际成员个数。

表 5-8　ENUM 类型

成员个数范围	字　节　数	成员个数范围	字　节　数
1≤L≤255	1	256≤L≤65535	2

说明：在定义字段时，若 ENUM 类型字段声明为 NULL，插入 NULL 为有效值，默认值为 NULL；若 ENUM 类型字段声明为 NOT NULL，插入 NULL 为无效值，默认值为 ENUM 类型成员的第 1 个成员。

4. SET 类型

SET 类型与 ENUM 类型十分相似，也是一个字符串对象。与 ENUM 类型不同的是，SET 类型一次可以选取多个成员，而 ENUM 类型则只能选取一个。当一个字符串设置字段值时，SET 类型可以取成员个数范围内的 0 个或多个值。SET 类型包含的成员个数和存储空间都不同，具体如表 5-9 所示，其中 L 表示实际成员个数。

表 5-9　SET 类型

成员个数范围	字　节　数	成员个数范围	字　节　数
1≤L≤8	1	25≤L≤32	4
9≤L≤16	2	33≤L≤64	8
17≤L≤24	3		

5.1.6　二进制字符串类型

在 MySQL 中，二进制字符串类型主要用于存储二进制数据，如图片、音频和视频等。MySQL 支持的二进制字符串类型主要包括 BIT、BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB 等，具体如表 5-10 所示，其中 M 和 L 都表示值的长度。

表 5-10　MySQL 二进制字符串类型

数 据 类 型	值 的 长 度	字　节　数
BIT(M)	1≤M≤64	约为(M+7)/8
BINARY(M)	M(0≤M≤255)	M
VARBINARY(M)	M(0≤M≤65535)	M+1
TINYBLOB	0≤L≤255	L+1
BLOB	0≤L≤65535(64KB)	L+2
MEDIUMBLOB	0≤L≤16777215(16MB)	L+3
LONGBLOB	0≤L≤4294967295(4GB)	L+4

1. BIT 类型

BIT 类型又称作位字段类型，主要存储二进制值，类似 010110。若没有指定长度 M，默认

为 1 位,表示只能存储 1 位二进制值。若分配的值的长度小于 M 位,则在值的左侧用 0 填充。

2. BINARY 类型与 VARBINARY 类型

BINARY 类型与 VARBINARY 类型主要用于存储二进制字符串。

BINARY(M)存储固定长度的二进制字符串,如果未指定长度 M,表示只能存储 1 字节。若存储字段不足 M 字节,将在右侧填充/0 以补齐指定长度;反之,超出的部分则会被截断。VARBINARY(M)存储可变长度的二进制字符串,必须指定 M,否则会报错。

3. BLOB 类型

在 MySQL 中,BLOB 类型包括 4 种类型,分别为 TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB 类型。BLOB 类型是一个二进制的对象,主要用于存储可变数量的数据,如图片、音频和视频等。

在实际开发中,该如何选择 BLOB 类型与 TEXT 类型? MySQL 中 BLOB 类型和 TEXT 类型在实际场景中的选择如下。

(1) BLOB 类型存储的是二进制字符串,而 TEXT 类型存储的是非二进制字符串。

(2) BLOB 类型的数据是以字节序列的形式存储的,因此在进行排序和比较时,会基于这些字节的数值进行操作。TEXT 类型的数据则是以字符序列的形式存储的,所以在排序和比较时,会根据字符集规则对这些字符进行操作。

(3) 在实际工作中,往往不会在 MySQL 数据库中使用 BLOB 类型存储大对象数据,通常会将图片、音频和视频文件存储到服务器的磁盘上,并将图片、音频和视频的访问路径存储到 MySQL 中。

5.2 创建数据表

5.2.1 约束概述

在 MySQL 中,约束是指对表中数据的一种限制,能够帮助数据库管理员更好地管理数据库,并且能够确保数据库中数据的完整性。数据完整性是指数据的精确性和可靠性,是防止数据库中存在不符合语义规定的数据和防止因错误信息的输入/输出造成无效操作或错误信息而提出的。例如,在数据表中存储身高值时,如果存入 300cm、400cm 这种无效的值就毫无意义了,所以使用约束限定表中的数据范围是很有必要的。

可以在创建数据表时执行 CREATE TABLE 语句规定约束,或者在数据表创建之后执行 ALTER TABLE 语句规定约束。

5.2.2 创建数据表的语法格式

数据表是数据库的重要组成部分,每个数据库都是由若干个数据表组成的。也就是说,没有数据表,就无法在数据库中存储数据。所以,在创建完数据库之后,接下来就要在创建好的数据库中创建新的数据表。创建数据表的过程是规定数据列属性的过程,同时也是实施数据完整性约束的过程。在 MySQL 中创建数据表的语法格式如下。

```
CREATE TABLE [IF NOT EXISTS] 表名称(
      字段 1 数据类型 [列级别约束条件] [默认值],
      字段 2 数据类型 [列级别约束条件] [默认值],
```

```
    …
    [表级别约束条件]
    );
```

说明：

(1) 表名称为需要创建的数据表的名称。

(2) 字段规定数据表中列的名称。

(3) 数据类型规定数据表中列的数据类型，如 VARCHAR、DATE 等。

(4) 列级别约束条件指定列级别字段的某些约束条件。

在 MySQL 中创建数据表的注意事项如下。

(1) 如果创建数据表时加上了 IF NOT EXISTS 关键字，则表示：若当前数据库中不存在要创建的数据表，则创建数据表；若当前数据库中已经存在要创建的数据表，则忽略建表语句，不再创建数据表。

(2) 在创建数据表时，还需要指定数据表中每列的名称和数据类型，多列之间需要以逗号进行分隔。

(3) 在 Windows 操作系统中，创建数据表的表名是不区分大小写的，但不能使用 SQL 中的关键字，如 INSERT、ALTER、DROP 等。

【案例 5-1】 创建数据表。

在数据库学习系统数据库(db_study)中创建一个数据表，名称为 tb_department，用于保存部门信息，分别给每个字段选择合适的数据类型，具体信息如表 5-11 所示。

表 5-11　部门表

字 段 名 称	数 据 类 型	描　　述
department_id	CHAR(3)	部门(X＋两位数字)
department_name	VARCHAR(50)	部门名称
department_phone	VARCHAR(13)	部门联系方式(11 位数字＋1～2 个间隔符)
department_address	VARCHAR(50)	部门所在地址

在 db_study 数据库中创建 tb_department 数据表前，需要使用“USE 数据库;”语句指定选择使用的数据库，再创建数据表，否则会报错。SQL 语句如下。

```
USE db_study;
CREATE TABLE tb_department
(
  department_id CHAR(3),
  department_name VARCHAR(50),
  department_phone VARCHAR(13),
  department_address VARCHAR(50)
);
```

执行上述 SQL 语句，结果如图 5-1 所示，表示创建 tb_department 数据表成功。

```
mysql> CREATE TABLE tb_department
    -> (
    -> department_id CHAR(3),
    -> department_name VARCHAR(50),
    -> department_phone VARCHAR(13),
    -> department_address VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

图 5-1　创建 tb_department 数据表

也可以使用 SHOW TABLES 语句查看数据表是否创建成功,如图 5-2 所示,tb_department 数据表已创建成功。

```
mysql> SHOW TABLES FROM db_study;
+-----------------+
| Tables_in_db_study |
+-----------------+
| tb_department   |
+-----------------+
1 row in set (0.00 sec)
```

图 5-2 tb_department 数据表创建成功

5.2.3 使用非空约束

非空约束(Not Null Constraint)是指数据表中某列的内容不允许为空,可以使用 NOT NULL 来表示。如果使用了非空约束,用户在添加数据时没有指定值,数据库系统会报错。非空约束的语法格式如下。

```
字段名 数据类型 NOT NULL;
```

说明:

(1) MySQL 默认所有类型的值都可以是 NULL。

(2) 只能某个列单独限定非空,不能组合非空。

(3) 空字符串('')不等于 NULL,0 也不等于 NULL。

【案例 5-2】 添加非空约束。

对案例 5-1 中的 tb_department 数据表进行完善,为 department_id 字段和 department_name 字段添加非空约束,其他字段则默认为空。当要对表结构进行修改时,则执行 DROP TABLE 语句先将数据表删除后再创建。SQL 语句如下。

```
DROP TABLE tb_department;
CREATE TABLE tb_department
(
  department_id CHAR(3) NOT NULL,
  department_name VARCHAR(50) NOT NULL,
  department_phone VARCHAR(13) NULL,
  department_address VARCHAR(50) NULL
);
DESC tb_department;
```

执行上述 SQL 语句,结果如图 5-3 所示。

```
mysql> DROP TABLE tb_department;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE tb_department
    -> (
    -> department_id CHAR(3) NOT NULL,
    -> department_name VARCHAR(50) NOT NULL,
    -> department_phone VARCHAR(13) NULL,
    -> department_address VARCHAR(50) NULL
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> DESC tb_department;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| department_id   | char(3)     | NO   |     | NULL    |       |
| department_name | varchar(50) | NO   |     | NULL    |       |
| department_phone| varchar(13) | YES  |     | NULL    |       |
| department_address | varchar(50) | YES |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-3 添加非空约束

由运行结果可知，当出现 Query OK 提示信息时，则表示成功删除或创建 tb_department 数据表。也可以执行“DESC 表名称;”语句查看数据表的结构，发现 department_id 和 department_name 字段已添加了非空约束。

说明：在实际工作中，任何开发，数据表一旦设计了就基本很难做修改。如果真的要进行表结构的修改，只有一个原则——删除表后重建。但需要给某个字段添加约束时，可以使用 ALTER 语句进行添加。

5.2.4　使用主键约束

在 MySQL 中创建数据表时，可以给数据表指定主键，主键又称为主码，是数据表中一列或多列的组合。主键约束(Primary Key Constraint)是使用最频繁的约束，主键约束既不能重复，也不能为空，主键能够唯一地标识数据表中的一条记录，可以结合外键定义不同数据表之间的关系，并且加快数据库查询的速度。可以使用 PRIMARY KEY 表示主键，简称 PK。

在创建数据表时设置主键约束，可以由一个字段组成，也可以多个字段联合组成。但不管使用哪种方法，一个数据表中只能设置一个主键。

1. 单列主键

单列主键只包含数据表中的一个字段。MySQL 中的单列主键不仅可以在定义列时同时指定，也可以在定义完所有列之后指定。指定单列主键的语法格式如下。

```
# 在定义列的同时指定主键
字段 数据类型 PRIMARY KEY [默认值]
# 在定义完所有列之后指定主键
[CONSTRAINT 约束名] PRIMARY KEY [字段名]
```

2. 多列联合主键

多列联合主键是支持多个字段共同组成的，只能在定义完所有列之后指定。指定多列联合主键的语法格式如下。

```
PRIMARY KEY [字段 1,字段 2,字段 3,…,字段 n]
```

【案例 5-3】 添加主键约束。

对案例 5-2 中的 tb_department 数据表进行完善，为 department_id 字段添加主键约束，SQL 语句如下。

```
DROP TABLE tb_department;
CREATE TABLE tb_department
(
  department_id CHAR(3) NOT NULL PRIMARY KEY,
  department_name VARCHAR(50) NOT NULL,
  department_phone VARCHAR(13) NULL,
  department_address VARCHAR(50) NULL
);
DESC tb_department;
```

执行上述 SQL 语句，结果如图 5-4 所示。

由运行结果可知，当出现 Query OK 提示信息时，则表示成功删除或创建 tb_department 数据表。也可以执行“DESC 表名称;”语句查看数据表的结构，发现 department_id 字段已添加了主键约束。

当数据表中不需要指定主键约束时，可以执行 DROP 语句将其删除，删除主键约束的语

```
mysql> DROP TABLE tb_department;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE tb_department
    -> (
    -> department_id  CHAR(3) NOT NULL PRIMARY KEY,
    -> department_name VARCHAR(50) NOT NULL,
    -> department_phone VARCHAR(13) NULL,
    -> department_address VARCHAR(50) NULL
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> DESC tb_department;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| department_id      | char(3)     | NO   | PRI | NULL    |       |
| department_name    | varchar(50) | NO   |     | NULL    |       |
| department_phone   | varchar(13) | YES  |     | NULL    |       |
| department_address | varchar(50) | YES  |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-4　添加主键约束

法格式如下。

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

说明：

(1) 表名称表示要删除主键约束的数据表的名称。

(2) PRIMARY KEY 为主键约束的关键字。

5.2.5　使用外键约束

外键约束(Foreign Key Constraint)用于在两个数据表的数据之间建立连接，可以是一列或多列。只要是数据表设计，一定要有外键关系，外键是作用在两个数据表上的约束，限定某个表的某个字段的引用完整性。一个表的外键可以为空，也可以不为空，当外键不为空时，则每个外键的值必须等于另一个表主键的某个值，一个表的外键可以不是本表的主键，但其对应另一个表的主键。在一个表中定义了外键之后，不允许删除另外一个表中具有关联关系的行数据。

外键是作用在两个表中，对于两个具有关联关系的表，又分为主表和从表。

主表(父表)：两个表具有关联关系时，关联字段中主键所在的表为主表。

从表(子表)：两个表具有关联关系时，关联字段中外键所在的表为从表。

指定外键约束的语法格式如下。

```
[CONSTRAINT 外键名] FOREIGN KEY 字段 1[,字段 2,字段 3,…]
REFERENCES 主表名 主键列 1[,主键 2,主键 3,…]
```

说明：

(1) 外键名定义外键约束的名称。

(2) 字段表示从表需要创建外键约束的字段列，可以由多列组成。

(3) 主表名为被从表外键所依赖的表的名称。

(4) 主键列为被应用的表中的列名，也可以由多列组成。

(5) CONSTRAINT 为创建约束的关键字。

(6) FROEIGN KEY 表示所创建约束的类型为外键约束。

(7) REFERENCES 表示被约束的列在主表中的某列。

【案例 5-4】 添加外键约束。

在 db_study 数据库中创建一个数据表，名称为 tb_class，用于保存班级信息，将班级号

(class_id)设置为主键,部门号(department_id)设置为外键,则班级表结构如表 5-12 所示。

表 5-12　班级表

字 段 名 称	数 据 类 型	NULL	约束	描　　述
class_id	CHAR(5)	否	主键	班级号(字母+4 位数字)
class_name	VARCHAR(50)	否	唯一	班级名称
department_id	CHAR(3)	是	外键	部门号(X+两位数字,与 tb_department 表的 department_id 数据保持一致)

在 db_study 数据库中创建 tb_class 数据表,按照要求添加外键约束的 SQL 语句如下。

```
CREATE TABLE tb_class
(
  class_id CHAR(5) NOT NULL PRIMARY KEY,
  class_name VARCHAR(50) NOT NULL,
  department_id CHAR(3) NULL,
  CONSTRAINT fk_department_id1 FOREIGN KEY(department_id)
REFERENCES tb_department(department_id)
);
DESC tb_class;
```

执行上述 SQL 语句,结果如图 5-5 所示。

```
mysql> CREATE TABLE tb_class
    -> (
    -> class_id CHAR(5) NOT NULL PRIMARY KEY,
    -> class_name VARCHAR(50) NOT NULL,
    -> department_id CHAR(3) NULL,
    -> CONSTRAINT fk_department_id1 FOREIGN KEY(department_id)
    -> REFERENCES tb_department(department_id)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> DESC tb_class;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   |     | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

图 5-5　添加外键约束

由运行结果可知,成功创建 tb_class 数据表后,可以执行“DESC 表名称;”语句查看数据表的结构,发现 tb_class 数据表中的 department_id 字段已经添加了外键约束。

说明:

(1) 因为主表是 tb_department,则必须先将其创建成功,才能创建 tb_class 数据表,由于案例 5-3 已创建成功,所以这里才能成功指定外键。

(2) 当需要删除数据表时,首先删除从表,再删除主表。

当数据表中不需要使用外键约束时,可以执行 DROP 语句将其删除,删除外键约束的语法格式如下。

```
ALTER TABLE 表名称
DROP FOREIGN KEY 字段名;
```

说明:

(1) 表名称为要删除的外键约束的数据表的名称。

(2) 字段名为需要删除字段的外键约束的名称。

(3) FOREIGN KEY 表示外键约束的关键字。

5.2.6 使用唯一约束

唯一约束(Unique Constraint)是指数据表中某列的数据不允许重复。例如,每个用户的E-mail地址不允许重复,就使用唯一性约束(UNIQUE)进行声明。唯一约束与主键约束相似的是它们都可以确保列的唯一性。不同的是,在一个表中可有多个唯一约束,并且设置唯一约束的列允许有空值,但是只能有一个空值;而在一个表中只能有一个主键约束,且不允许有空值。唯一约束通常设置在除了主键以外的其他列上,语法格式如下。

```
字段名 数据类型 UNIQUE
```

说明:UNIQUE是唯一约束的关键字。

【案例5-5】 添加唯一约束。

为了避免班级名称重复,需要为其添加唯一约束。对案例5-4中的tb_class数据表进行完善,将班级名称(class_name)设置为唯一约束,SQL语句如下。

```
DROP TABLE tb_class;
CREATE TABLE tb_class
(
  class_id CHAR(5) NOT NULL PRIMARY KEY,
  class_name VARCHAR(50) NOT NULL UNIQUE,
  department_id CHAR(3) NULL,
  CONSTRAINT fk_department_id1 FOREIGN KEY(department_id)
REFERENCES tb_department(department_id)
);
DESC tb_class;
```

执行上述SQL语句,结果如图5-6所示。

```
mysql> DROP TABLE tb_class;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE tb_class
    -> (
    -> class_id CHAR(5) NOT NULL PRIMARY KEY,
    -> class_name VARCHAR(50) NOT NULL UNIQUE,
    -> department_id CHAR(3) NULL,
    -> CONSTRAINT fk_department_id1 FOREIGN KEY(department_id)
    -> REFERENCES tb_department(department_id)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> DESC tb_class;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

图5-6 添加唯一约束

由运行结果可知,成功创建tb_class数据表后,可以执行"DESC 表名称;"语句查看数据表的结构,发现tb_class数据表中的class_name字段已经添加了唯一约束。

说明:

(1) 同一个表中可以有某列的值唯一,也可以是多列组合的值唯一。

(2) 唯一约束允许列值为空。

(3) 在创建唯一约束时,如果不给唯一约束命名,就默认和列名相同。

5.2.7 使用自增列

在数据库应用中,需要系统在每次插入记录时能自动生成字段的主键值,可以通过为表主

键添加 AUTO_INCREMENT 关键字实现，意为自增长。当主键定义为自增长后，这个主键的值就不再需要用户输入数据，而由数据库根据定义自动赋值。每增加一条记录，主键会自动以相同的步长进行增长。设置自增列的语法格式如下。

```
字段名 数据类型 AUTO_INCREMENT
```

【案例 5-6】 添加自增列。

在数据库学习系统数据库(db_study)中创建一个数据表，名称为 tb_student，用于保存学生信息，其中需要将 student_id 字段(学号)设置为自增长，初始值为 20220101001。学生表结构如表 5-13 所示。

表 5-13　学生表

字 段 名 称	数 据 类 型	NULL	约束	描　　述
student_id	BIGINT(11)	否	主键	学号(自增长，初始值为 20220101001，每次加 1)
student_name	VARCHAR(20)	否		姓名
student_gender	ENUM	是		性别('男','女')
student_height	SMALLINT(3)	是		身高(无符号整数，范围为 0～255)
student_birthday	TIMESTAMP	是		出生日期
class_id	CHAR(5)	是	外键	班号(B+4 位数字，与 tb_class 数据表中 class_id 数据保持一致)
student_phone	CHAR(13)	是	唯一	联系电话(13 位，中间有两个-分隔符)

在 db_study 数据库中创建 tb_student 数据表，按照要求添加自增列的 SQL 语句如下。

```
CREATE TABLE tb_student
(
  student_id BIGINT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  student_name VARCHAR(20) NOT NULL,
  student_gender ENUM('男','女') NULL,
  student_height TINYINT(3) UNSIGNED NULL,
  student_birthday TIMESTAMP NULL,
  class_id CHAR(5) NULL,
  student_phone CHAR(13) NULL UNIQUE,
  CONSTRAINT fk_class_id1 FOREIGN KEY(class_id) REFERENCES tb_class(class_id )
)
auto_increment = 20220101001;
DESC tb_student;
```

执行上述 SQL 语句，结果如图 5-7 所示。

由运行结果可知，成功创建 tb_student 数据表后，可以执行“DESC 表名称;”语句查看数据表的结构，发现在 tb_student 表中已将 student_id 字段设置为自增列。

说明：

(1) 在默认情况下，自增列的初始值为 1，每新增一条记录，字段值自动加 1。

(2) 一个表中只能有一个字段使用自增列，且该字段必须为主键或主键的一部分。

(3) 自增列只能是整数类型，如 TINYINT、SMALLINT、INT、BIGINT 等。

5.2.8　使用默认值约束

默认值约束(Default Constraint)是给某个字段/某列指定默认值，一旦设置默认值，在插

```
mysql> CREATE TABLE tb_student
    -> (
    -> student_id BIGINT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> student_name VARCHAR(20) NOT NULL,
    -> student_gender ENUM('男','女') NULL,
    -> student_height TINYINT(3) UNSIGNED NULL,
    -> student_birthday TIMESTAMP NULL,
    -> class_id CHAR(5) NULL,
    -> student_phone CHAR(13) NULL UNIQUE,
    -> CONSTRAINT fk_class_id1 FOREIGN KEY(class_id) REFERENCES tb_class(class_id )
    -> )
    -> auto_increment=20220101001;
Query OK, 0 rows affected, 2 warnings (0.03 sec)

mysql> DESC tb_student;
+------------------+---------------------+------+-----+---------+----------------+
| Field            | Type                | Null | Key | Default | Extra          |
+------------------+---------------------+------+-----+---------+----------------+
| student_id       | bigint              | NO   | PRI | NULL    | auto_increment |
| student_name     | varchar(20)         | NO   |     | NULL    |                |
| student_gender   | enum('男','女')     | YES  |     | NULL    |                |
| student_height   | tinyint unsigned    | YES  |     | NULL    |                |
| student_birthday | timestamp           | YES  |     | NULL    |                |
| class_id         | char(5)             | YES  | MUL | NULL    |                |
| student_phone    | char(13)            | YES  | UNI | NULL    |                |
+------------------+---------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

图 5-7　添加自增列

入数据时,如果此字段没有显式赋值,则赋值为默认值。例如,1 班中的学生,那么班级编号就可以指定为默认值 01。如果插入一条新的记录并且没有为这个字段赋值,则系统会自动为班级编号这个字段赋值为 01。默认值约束的语法格式如下。

```
字段名 数据类型 DEFAULT 默认值
```

说明:

(1) DEFAULT 表示默认值约束的关键字。

(2) 默认值是一个具体的值,也可以是通过表达式得到的一个值,但必须与该字段的数据类型相匹配。

(3) 一个表可以有很多默认值约束。在创建表时为列添加默认值,可以一次为多个列添加默认值,需要注意不同列的数据类型。

(4) 默认值约束意味着如果该字段没有手动赋值,会按默认值处理。

5.2.9　使用检查约束

检查约束(Check Constraint)是指在进行数据更新前设置一些过滤条件,满足此条件的数据可以实现更新。可以使用 CHECK 关键字定义检查约束,用于检验输入值,拒绝接受不满足条件的值,减少无效数据的输入。检查约束的语法格式如下。

```
CHECK(检查约束的条件)
```

说明: CHECK 表示设置检查约束的关键字。

虽然有检查约束这种概念,但是在实际场景中,会比较少使用检查约束,为什么? 因为所有检查约束都是逐个进行过滤,如果在一个数据表中进行了过多的检查约束,在进行数据更新时会严重影响程序的性能。

5.2.10　查看数据表结构

在 MySQL 中使用 SQL 语句创建数据表之后,可以查看数据表结构,确认数据表的定义是否正确。MySQL 支持使用 DESCRIBE/DESC 语句查看数据表结构,也支持使用 SHOW CREATE TABLE 语句查看数据表结构。下面分别介绍这两种查看数据表结构的方法。

1. DESCRIBE/DESC 语句

使用 DESCRIBE/DESC 语句可以查看数据表的基本结构，语法格式如下。

```
DESCRIBE 表名称;
```

或

```
DESC 表名称;
```

说明：表名称为需要查看数据表结构的表的名称。

【案例 5-7】 查看数据表的基本结构。

使用 DESCRIBE/DESC 语句查看 tb_class 数据表的基本结构，SQL 语句如下。

```
DESCRIBE tb_class;
```

或

```
DESC tb_class;
```

执行上述 SQL 语句，结果如图 5-8 所示。

```
mysql> DESCRIBE tb_class;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> DESC tb_class;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

图 5-8　查看数据表的基本结构

由运行结果可知，使用 DESCRIBE 和 DESC 语句的查询结果相同，可以查看表的字段名称、字段数据类型、是否为主键、是否唯一等。其中，各个字段的含义分别如下。

(1) Field：字段名称。

(2) Type：字段类型，这里的 CHAR、VARCHAR 是文本字符串类型。

(3) Null：表示该列是否可以存储 NULL 值。

(4) Key：表示该列是否已编制索引。PRI 表示该列是表主键的一部分；UNI 表示该列是 UNIQUE 索引的一部分；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，给出默认值。

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

2. SHOW CREATE TABLE 语句

使用 SHOW CREATE TABLE 语句不仅可以查看表创建时的详细语句，还可以查看存储引擎和字符编码，语法格式如下。

```
SHOW CREATE TABLE 表名称;
```

或

```
SHOW CREATE TABLE 表名称\G
```

【案例 5-8】 查看数据表的详细信息。

使用 SHOW CREATE TABLE 语句查看 tb_class 数据表的详细信息,SQL 语句如下。

```
SHOW CREATE TABLE tb_class;
```

或

```
SHOW CREATE TABLE tb_class\G
```

执行上述 SQL 语句,结果如图 5-9 和图 5-10 所示。

```
mysql> SHOW CREATE TABLE tb_class;
+----------+--------------------------------------------------------------------------+
| Table    | Create Table                                                             |
+----------+--------------------------------------------------------------------------+
| tb_class | CREATE TABLE `tb_class` (
  `class_id` char(5) NOT NULL,
  `class_name` varchar(50) NOT NULL,
  `department_id` char(3) DEFAULT NULL,
  PRIMARY KEY (`class_id`),
  UNIQUE KEY `class_name` (`class_name`),
  KEY `fk_department_id1` (`department_id`),
  CONSTRAINT `fk_department_id1` FOREIGN KEY (`department_id`) REFERENCES `tb_department` (`department_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+----------+--------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

图 5-9 查看数据表的详细信息(1)

```
mysql> SHOW CREATE TABLE tb_class\G
*************************** 1. row ***************************
       Table: tb_class
Create Table: CREATE TABLE `tb_class` (
  `class_id` char(5) NOT NULL,
  `class_name` varchar(50) NOT NULL,
  `department_id` char(3) DEFAULT NULL,
  PRIMARY KEY (`class_id`),
  UNIQUE KEY `class_name` (`class_name`),
  KEY `fk_department_id1` (`department_id`),
  CONSTRAINT `fk_department_id1` FOREIGN KEY (`department_id`) REFERENCES `tb_department` (`department_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

图 5-10 查看数据表的详细信息(2)

由运行结果可知,执行"SHOW CREATE TABLE 表名称;"和"SHOW CREATE TABLE 表名称\G"语句查看数据表的详细信息的结果是相同的。\G 参数相当于格式化输出,使用\G 之后,可以看到输出内容具有较高的易读性。

5.3 修改数据表

5.3.1 重命名数据表

在实际开发中,还需要根据实际情况对数据表进行修改,当需要修改数据表的名称时,则可以执行 ALTER TABLE 语句实现,具体语法格式如下。其中,[TO]表示可选参数,使用与否不影响执行结果。

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

【案例 5-9】 重命名数据表。

在已有的 tb_class 数据表中,将 tb_class 重命名为"班级表",使用 ALTER TABLE 语句

修改新的表名称，SQL 语句如下。

```
ALTER TABLE tb_class RENAME 班级表;
```

执行上述 SQL 语句，再执行“SHOW TABLES;”语句查看数据表，发现 tb_class 数据表已成功重命名为“班级表”，结果如图 5-11 所示。

```
mysql> ALTER TABLE tb_class RENAME 班级表;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_db_study |
+--------------------+
| tb_department      |
| tb_student         |
| 班级表             |
+--------------------+
3 rows in set (0.00 sec)
```

图 5-11　重命名 tb_class 数据表

5.3.2　添加字段

在实际工作中，随着业务需求的变化，可能需要在表中添加新字段，添加字段时可以修改字段的排列位置。在 MySQL 中添加新字段的语法格式如下。

```
ALTER TABLE 表名称 ADD [COLUMN] 新字段名 字段类型 [FIRST|AFTER 已存在的字段名];
```

说明：

(1) FIRST 是可选参数，其作用是将新添加的字段设置为表的第 1 个字段。

(2) AFTER 是可选参数，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。

【案例 5-10】 添加新的字段。

对案例 5-9 中的班级表进行完善。为了统计每个班的总人数，现在需要在班级表中添加新的字段，并命名为 class_size，数据类型为 TINYINT，SQL 语句如下。

```
ALTER TABLE 班级表 ADD class_size TINYINT(2);
```

执行上述 SQL 语句，再执行“DESC 表名称;”语句查看数据表结构，发现 class_size 字段已添加到班级表中，如图 5-12 所示。

```
mysql> ALTER TABLE 班级表 ADD class_size TINYINT(2);
Query OK, 0 rows affected, 1 warning (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> DESC 班级表;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
| class_size    | tinyint     | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-12　在班级表中添加新字段

5.3.3　修改字段

在 MySQL 中，创建好数据表后，可以使用 ALTER TABLE 语句修改字段的数据类型。语法格式如下。

```
ALTER TABLE 表名称 MODIFY 字段名 字段类型;
```

说明:

(1) 表名称为需要修改数据表的名称。

(2) 字段名表示需要添加数据类型的字段列。

(3) 字段类型表示该字段需要修改的数据类型。

如果在创建数据表时没有添加约束,也可以执行 ALTER TABLE 语句进行添加或修改,不同的约束有不同的修改方式。下面分别介绍在修改数据表字段时添加约束的语法格式。

1. 修改数据表字段时添加主键约束

创建完数据表后,如果还需要为数据表的某个字段添加主键约束,可以不重新创建数据表,使用 ALTER 语句为现有的数据表添加主键,语法格式如下。

```
ALTER TABLE 表名称
ADD CONSTRAINT 约束名 PRIMARY KEY [字段 1,字段 2,字段 3,…,字段 n];
```

说明:

(1) 约束名表示需要添加外键约束的名称。

(2) 字段表示需要添加外键约束的字段列,可以由多个列组成。

(3) CONSTRAINT 表示需要创建约束的关键字。

(4) PRIMARY KEY 表示所添加约束的类型为主键约束。

2. 修改数据表字段时添加外键约束

如果在创建数据表时没有创建外键,可以使用 ALTER 语句为现有的数据表添加外键,语法格式如下。

```
ALTER TABLE 表名称
ADD CONSTRAINT 约束名 FOREIGN KEY [字段 1,字段 2,字段 3,…,字段 n] REFERENCES 主表名 主键列 1
[,主键 2,主键 3,…];
```

说明:

(1) 约束名表示需要添加的外键约束名称。

(2) CONSTRAINT 表示需要添加约束的关键字。

(3) FOREIGN KEY 表示所添加约束的类型为外键约束。

(4) 主键列表示需要被应用的表中的列名,也可以由多个列组成。

3. 修改数据表字段时添加唯一约束

如果在创建数据表时没有创建唯一约束,可以使用 ALTER 语句为现有的数据表添加唯一约束,但是需要保证添加唯一约束的列中存储的值没有重复的。语法格式如下。

```
ALTER TABLE 表名称 ADD CONSTRAINT 约束名 UNIQUE(字段名);
```

说明:

(1) 约束名表示需要添加的唯一约束名称。

(2) 字段名表示需要设置唯一约束的字段名称。

(3) UNIQUE 表示唯一约束的关键字。

4. 修改数据表字段时添加自增列

如果在创建数据表时没有创建自增列,可以使用 ALTER 语句为现有的数据表添加自增列,语法格式如下。

```
ALTER TABLE 表名称 CHANGE 字段名 数据类型 UNSIGNED AUTO_INCREMENT;
```

说明：

(1) CHANGE 表示修改列属性的关键字。

(2) UNSIGNED 表示需要自增长的数值无符号化。

(3) AUTO_INCREMENT 表示自增列约束的关键字。

5. 修改数据表字段时添加默认值约束

如果在创建数据表时没有创建默认值约束，可以使用 ALTER 语句为现有的数据表添加默认值约束，语法格式如下。

```
ALTER TABLE 表名称 ALTER 约束名 SET DEFAULT 默认值;
```

说明：

(1) 约束名表示添加默认值的约束名。

(2) 默认值为具体的一个值或通过表达式得到的一个值，但该值必须与该字段的数据类型相匹配。

【案例 5-11】 修改字段的约束条件。

在添加 class_size 字段时，没有添加唯一约束，现在为班级表中的 class_size 字段添加唯一约束，SQL 语句如下。

```
ALTER TABLE 班级表 ADD CONSTRAINT uq_class_size1 UNIQUE(class_size);
```

执行上述 SQL 语句，再执行“DESC 表名称;”语句查看字段的约束是否修改成功，发现班级表中的 class_size 字段已添加唯一约束，如图 5-13 所示。

```
mysql> ALTER TABLE 班级表 ADD CONSTRAINT uq_class_size1 UNIQUE(class_size);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC 班级表;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
| class_size    | tinyint     | YES  | UNI | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-13　修改字段的约束条件

5.3.4　重命名字段

重命名字段就是把旧的字段名修改为一个新的字段名，语法格式如下。

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

说明：

(1) 旧字段名为修改前的字段名称。

(2) 新字段名为修改后的字段名称。

(3) 新数据类型表示修改后的数据类型，如果不需要修改字段的数据类型，将新数据类型设置为与原来一样即可，但数据类型不能为空。

【案例 5-12】 修改字段名称。

将班级表中的 class_size 字段的名称修改为“班级人数”，数据类型不变，SQL 语句如下。

```
ALTER TABLE 班级表 CHANGE class_size 班级人数 TINYINT(2);
```

执行上述 SQL 语句，再执行“DESC 表名称;”语句查看字段名称是否修改成功，发现班级

表中的 class_size 字段已成功修改成新的字段名称“班级人数”,如图 5-14 所示。

```
mysql> ALTER TABLE 班级表 CHANGE class_size 班级人数 TINYINT(2);
Query OK, 0 rows affected, 1 warning (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> DESC 班级表;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
| 班级人数      | tinyint     | YES  | UNI | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-14　修改字段名称

5.3.5　修改字段的排列位置

对于一个数据表,在创建时,字段就在表中的排列顺序就已经确定了,但表的结构并不是完全不可以改变的,也可以执行 ALTER TABLE 语句改变表中字段的位置,语法格式如下。

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

说明:

(1) MODIFY 表示修改列属性的关键字。

(2) 字段 1 表示需要修改位置的字段。

(3) 数据类型为字段 1 的数据类型。

(4) 字段 2 表示需要插入新字段的前一个字段。

(5) FIRST 的作用是将字段 1 修改为数据表中的第 1 个字段。

(6) AFTER 的作用是将字段 1 插到字段 2 的后面。

【案例 5-13】 修改字段的排列位置。

将班级表中的“班级人数”字段的位置排列到 class_name 字段的后面,SQL 语句如下。

```
ALTER TABLE 班级表 MODIFY 班级人数 TINYINT(2) AFTER class_name;
```

执行上述 SQL 语句,再执行“DESC 表名称;”语句查看数据表结构,发现班级表中“班级人数”字段已排列在 class_name 字段的后面,如图 5-15 所示。

```
mysql> ALTER TABLE 班级表 MODIFY 班级人数 TINYINT(2) AFTER class_name;
Query OK, 0 rows affected, 1 warning (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> DESC 班级表;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| class_id      | char(5)     | NO   | PRI | NULL    |       |
| class_name    | varchar(50) | NO   | UNI | NULL    |       |
| 班级人数      | tinyint     | YES  | UNI | NULL    |       |
| department_id | char(3)     | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

图 5-15　修改字段的排列位置

5.3.6　删除字段

在 MySQL 中,删除字段就是将数据表中的某个字段从表中移除,语法格式如下。

```
ALTER TABLE 表名称 DROP 字段名;
```

【案例 5-14】 删除字段。

在班级表中,将刚修改的“班级人数”字段删除,SQL 语句如下。

```
ALTER TABLE 班级表 DROP 班级人数;
```

执行上述 SQL 语句，再执行“DESC 表名称；”语句查看数据表结构，发现班级表中的“班级人数”字段已删除成功，结果如图 5-16 所示。

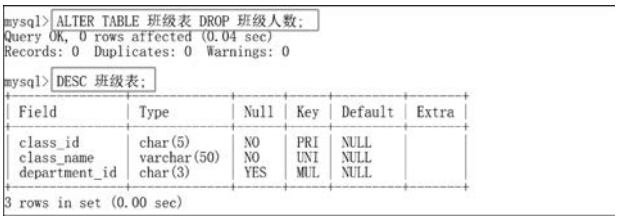

图 5-16　删除字段

5.4　删除数据表

当一个数据表不再被需要时，可以将其删除。但是，在删除表的同时，数据表的结构和表中的所有数据都会被删除，所以在删除数据表前最好先做好备份，以免造成无法弥补的损失。在 MySQL 中删除数据表有两种情况，一种是删除没有被关联的表，另外一种是删除被其他数据表关联的数据表。下面分别介绍这两种情况。

5.4.1　删除没有被关联的数据表

使用 DROP TABLE 语句可以一次删除一个或多个没有被其他数据表关联的数据表，语法格式如下。

```
DROP TABLE [IF EXISTS] 数据表 1 [,数据表 2,数据表 3,…,数据表 n];
```

说明：

(1) 可以同时删除多个数据表，相互之间用逗号隔开即可。

(2) IF EXISTS 用于在删除前判断表是否存在。

【案例 5-15】 删除没有被关联的数据表。

在 db_study 数据库中，先创建一个没有关联其他数据表的数据表，称为课程表(tb_course)，用于保存课程信息，其中课程号(course_id)设置为主键，课程名称(course_name)不可以重复。课程表结构如表 5-14 所示。

表 5-14　课程表

字段名称	数据类型	NULL	约束	描　述
course_id	CHAR(5)	否	主键	课程号(K+4 位数字)
course_name	VARCHAR(50)	否	唯一	课程名称
course_type	ENUM	是		课程类型(公共必修课、公共选修课、专业基础课、专业选修课、集中实践课、拓展课)
course_credit	TINYINT(3)	是		课程学分(无符号整数，范围为 0～255)
course_describe	TEXT	是		课程描述(课程介绍)

在 db_study 数据库中创建 tb_course 数据表，SQL 语句如下。

```
CREATE TABLE tb_course
(
```

```
  course_id CHAR(5) NOT NULL PRIMARY KEY,
  course_name VARCHAR(50) NOT NULL UNIQUE,
  course_type ENUM('公共必修课','公共选修课','专业基础课','专业选修课','集中实践课','拓展课')
NULL,
  course_credit TINYINT(3) UNSIGNED NULL,
  course_describe TEXT NULL
);
SHOW TABLES;
```

执行上述 SQL 语句,再执行“SHOW TABLES;”语句查看课程表是否创建成功,结果如图 5-17 所示。

```
mysql> CREATE TABLE tb_course
    -> (
    -> course_id CHAR(5) NOT NULL PRIMARY KEY,
    -> course_name VARCHAR(50) NOT NULL UNIQUE,
    -> course_type ENUM('公共必修课','公共选修课','专业基础课','专业选修课','集中实践课','拓展
课') NULL,
    -> course_credit TINYINT(3) UNSIGNED NULL,
    -> course_describe TEXT NULL
    -> );
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_db_study |
+--------------------+
| tb_course          |
| tb_department      |
| tb_student         |
| 班级表             |
+--------------------+
4 rows in set (0.00 sec)
```

图 5-17　创建 tb_course 数据表

由运行结果可知,tb_course 数据表已创建成功。在 db_study 数据库中,tb_course 数据表与其他数据表没有任何关联,当想要删除 tb_course 数据表时,可以直接使用以下 SQL 语句。

```
DROP TABLE IF EXISTS tb_course;
```

执行上述 SQL 语句,再执行“SHOW TABLES;”语句查看 tb_course 数据表是否删除成功,发现 db_study 数据库中已经没有 tb_course 数据表了,说明已删除成功,如图 5-18 所示。

```
mysql> DROP TABLE IF EXISTS tb_course;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_db_study |
+--------------------+
| tb_department      |
| tb_student         |
| 班级表             |
+--------------------+
3 rows in set (0.00 sec)
```

图 5-18　删除 tb_course 数据表

5.4.2　删除被其他数据表关联的数据表

在数据表之间存在外键关联的情况下,如果直接删除父表,会显示删除失败,原因是直接删除将破坏表的完整性。如果必须要删除,可以先直接删除与它关联的子表,再删除父表,这样就同时删除了两个数据表中的数据。或者将关联表的外键约束取消,再删除父表,适用于需要保留子表的数据,只删除父表的情况。

在 MySQL 中删除外键约束的语法格式如下。

```
ALTER TABLE 表名称 DROP FOREIGN KEY 外键约束名;
```

【案例 5-16】 删除被其他数据表关联的数据表。

在 db_study 数据库中，将 tb_department 数据表删除，但在案例 5-4 中已经将 department_id 字段设置为外键，如果直接删除 tb_department 数据表，会显示失败。SQL 语句如下。

```
DROP TABLE IF EXISTS tb_department;
```

执行上述 SQL 语句，可以看到直接删除 tb_department 主表时，MySQL 会报错，如图 5-19 所示。

```
mysql> DROP TABLE IF EXISTS tb_department;
ERROR 3730 (HY000): Cannot drop table 'tb_department' referenced by a foreign key constraint 'f
k_department_id1' on table '班级表'.
mysql>
```

图 5-19　直接删除 tb_department 数据表

由错误提示信息可知，department_id 是班级表的外键约束字段，班级表为子表，具有名称为 fk_department_id1 的外键约束；tb_department 为父表，其主键 department_id 被子表班级表所关联。需要解除关联子表班级表的外键约束，SQL 语句如下。

```
ALTER TABLE 班级表 DROP FOREIGN KEY fk_department_id1;
```

执行上述 SQL 语句，结果如图 5-20 所示。

```
mysql> ALTER TABLE 班级表 DROP FOREIGN KEY fk_department_id1;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

图 5-20　解除关联子表的外键约束

由运行结果可知，出现 Query OK 提示信息，说明已经将关联子表的外键约束删除，则可以将父表 tb_department 删除，SQL 语句如下。

```
DROP TABLE IF EXISTS tb_department;
```

执行上述 SQL 语句，再执行“SHOW TABLES;”语句查看数据库结构，发现数据库中已经没有 tb_department 数据表了，说明已删除成功，如图 5-21 所示。

```
mysql> DROP TABLE IF EXISTS tb_department;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_db_study |
+----------------+
| tb_student     |
| 班级表         |
+----------------+
2 rows in set (0.00 sec)
```

图 5-21　删除被关联的主表

课业任务

扫一扫

视频讲解

*课业任务 5-1　创建用户登录表

【能力测试点】

创建数据表。

【任务实现步骤】

任务需求：在 db_study 数据库中创建一个用户登录表(tb_login)，由序号(login_id)、用户名(login_name)、用户密码(login_password)3 个字段组成，其中序号为自增长，初始值为 1，每增加一条记录加 1，用户名设置唯一约束。用户登录表结构如表 5-15 所示。

表 5-15　用户登录表

字段名称	数据类型	NULL	约束	描　述
login_id	INT(5)	否	主键	序号(自增长,初始值为 1,每次加 1)
login_name	VARCHAR(20)	否	唯一	用户名
login_password	VARCHAR(45)	否		用户密码

(1) 按任务需求创建用户登录表的 SQL 语句如下。

```
CREATE TABLE tb_login
(
  login_id INT(5) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  login_name VARCHAR(20) NOT NULL UNIQUE,
  login_password VARCHAR(45) NOT NULL
);
```

(2) 执行上述 SQL 语句,结果如图 5-22 所示。

(3) 执行"SHOW TABLES;"语句查看数据库中所有表,结果如图 5-23 所示。由运行结果可知 tb_login 数据表创建成功。

```
mysql> CREATE TABLE tb_login
    -> (
    -> login_id INT(5) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> login_name VARCHAR(20) NOT NULL UNIQUE,
    -> login_password VARCHAR(45) NOT NULL
    -> );
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

图 5-22　创建用户登录表

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_db_study |
+-----------------+
| tb_login        |
| tb_student      |
| 班级表          |
+-----------------+
3 rows in set (0.00 sec)
```

图 5-23　tb_login 数据表创建成功

扫一扫

视频讲解

课业任务 5-2　向用户登录表中添加字段

【能力测试点】

向数据表中添加字段。

【任务实现步骤】

任务需求:课业任务 5-1 已经创建 tb_login 数据表,向表中添加一个备注字段(login_remark),数据类型为 VARCHAR。

(1) 按任务需求在用户登录表中添加字段的 SQL 语句如下。

```
ALTER TABLE tb_login ADD login_remark VARCHAR(255);
```

(2) 执行上述 SQL 语句,结果如图 5-24 所示。

(3) 执行"DESC 表名称;"语句查看数据表结构,结果如图 5-25 所示。由运行结果可知 login_remark 字段已存在,说明字段添加成功。

```
mysql> ALTER TABLE tb_login ADD login_remark VARCHAR(255);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

图 5-24　添加 login_remark 字段

```
mysql> DESC tb_login;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| login_id       | int          | NO   | PRI | NULL    | auto_increment |
| login_name     | varchar(20)  | NO   | UNI | NULL    |                |
| login_password | varchar(45)  | NO   |     | NULL    |                |
| login_remark   | varchar(255) | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

图 5-25　login_remark 字段添加成功

扫一扫

视频讲解

课业任务 5-3　修改用户登录表中字段的数据类型

【能力测试点】

修改字段数据类型。

【任务实现步骤】

任务需求：将 tb_login 数据表的 login_remark 字段修改为 TEXT 类型。

(1) 按任务需求修改用户登录表中字段类型的 SQL 语句如下。

```
ALTER TABLE tb_login MODIFY login_remark TEXT;
```

(2) 执行上述 SQL 语句，结果如图 5-26 所示。

```
mysql> ALTER TABLE tb_login MODIFY login_remark TEXT;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

图 5-26　修改 login_remark 字段的数据类型

(3) 执行“DESC 表名称;”语句查看数据表信息，结果如图 5-27 所示。由运行结果可知 login_remark 字段为 TEXT 类型，说明 login_remark 字段的数据类型已修改成功。

```
mysql> DESC tb_login;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| login_id       | int         | NO   | PRI | NULL    | auto_increment |
| login_name     | varchar(20) | NO   | UNI | NULL    |                |
| login_password | varchar(45) | NO   |     | NULL    |                |
| login_remark   | text        | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

图 5-27　login_remark 字段数据类型修改成功

扫一扫

视频讲解

课业任务 5-4　删除用户登录表中的一个字段

【能力测试点】

删除数据表中的字段。

【任务实现步骤】

任务需求：在 db_study 数据库中，删除 tb_login 数据表中的 login_remark 字段。

(1) 按任务需求删除用户登录表中备注字段的 SQL 语句如下。

```
ALTER TABLE tb_login DROP login_remark;
```

(2) 执行上述 SQL 语句，删除 login_remark 字段，结果如图 5-28 所示。

(3) 执行“DESC 表名称;”语句查看备注字段是否删除成功，结果如图 5-29 所示。由运行结果可知，tb_login 数据表中没有 login_remark 字段，即说明该字段删除成功。

```
mysql> ALTER TABLE tb_login DROP login_remark;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

图 5-28　删除 login_remark 字段

```
mysql> DESC tb_login;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| login_id       | int         | NO   | PRI | NULL    | auto_increment |
| login_name     | varchar(20) | NO   | UNI | NULL    |                |
| login_password | varchar(45) | NO   |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

图 5-29　login_remark 字段删除成功

扫一扫

视频讲解

课业任务 5-5　删除用户登录表

【能力测试点】

删除数据表。

【任务实现步骤】

(1) 当不需要用到用户登录表时，可以将其删除。由于在 db_study 数据库中，tb_login 数据表与其他数据表没有关联，则可以直接使用 DROP TABLE 语句进行删除，SQL 语句如下。

```
DROP TABLE tb_login;
```

(2) 执行上述 SQL 语句,成功删除 tb_login 数据表,结果如图 5-30 所示。

(3) 最后执行"SHOW TABLES;"语句查看列表中是否还有 tb_login 数据表,结果如图 5-31 所示。由运行结果可知,db_study 数据库中已没有 tb_login 数据表,即说明该数据表删除成功。

```
mysql> DROP TABLE tb_login;
Query OK, 0 rows affected (0.01 sec)
```

图 5-30 删除 tb_login 数据表

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_db_study |
+---------------+
| tb_student    |
| 班级表        |
+---------------+
2 rows in set (0.00 sec)
```

图 5-31 tb_login 数据表删除成功

扫一扫

视频讲解

课业任务 5-6 使用 MySQL Workbench 工具创建用户登录表

【能力测试点】

使用数据库图形化管理工具 MySQL Workbench 创建数据表。

【任务实现步骤】

任务需求:使用数据库图形化管理工具 MySQL Workbench 在 db_study 数据库中创建用户登录表。

(1) 启动 MySQL Workbench,登录成功后,在界面左侧的数据库对象窗口中展开 db_study 数据库,右击 Tables 选项,在弹出的快捷菜单中选择 Create Table(创建数据表)菜单命令,如图 5-32 所示。

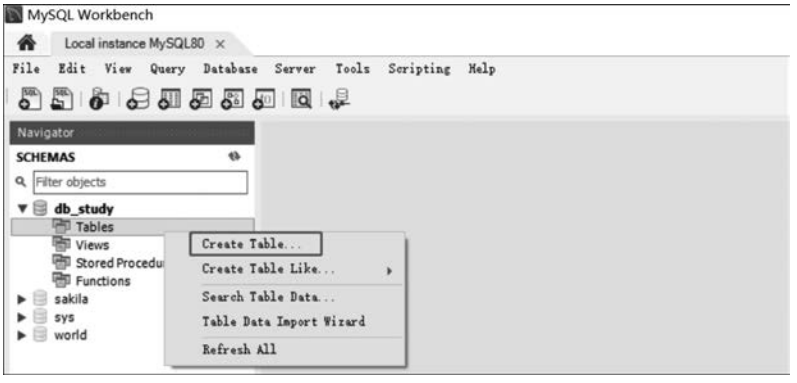

图 5-32 创建数据表

(2) 在弹出的 tb_login-Table 窗口中可以设置用户登录表信息,具体的表结构信息如表 5-15 所示。设置完数据表的基本信息后单击 Apply(确认)按钮,如图 5-33 所示。

(3) 弹出一个确定对话框,显示创建 tb_login 数据表的 SQL 语句,确认无误后,单击 Apply(确认)按钮完成 tb_login 数据表的创建。

(4) 在弹出的对话框中单击 Finish 按钮,即可完成创建用户登录表的操作。

(5) 回到主界面,可以看到 tb_login 数据表已经创建成功。

扫一扫

视频讲解

课业任务 5-7 使用 Navicat Premium 工具向用户登录表添加字段

【能力测试点】

使用图形化管理工具 Navicat Premium 添加字段。

图 5-33　tb_login 数据表信息

【任务实现步骤】

任务需求：由于在课业任务 5-6 中已经创建了用户登录表，所以本任务使用图形化管理工具 Navicat Premium 在 db_study 数据库中向 tb_login 数据表添加一个备注字段 login_remark，数据类型为 TEXT 类型，默认为空。

(1) 启动 Navicat Premiun 16，登录成功后，右击用户登录表 tb_login，在弹出的快捷菜单中选择“设计表”，如图 5-34 所示。

图 5-34　选择“设计表”

(2) 在弹出的窗口中单击“添加字段”按钮，输入新字段名 login_remark，数据类型设置为 text，字符集选择 utf8mb4，排序规则选择 utf8mb4_general_ci，如图 5-35 所示。

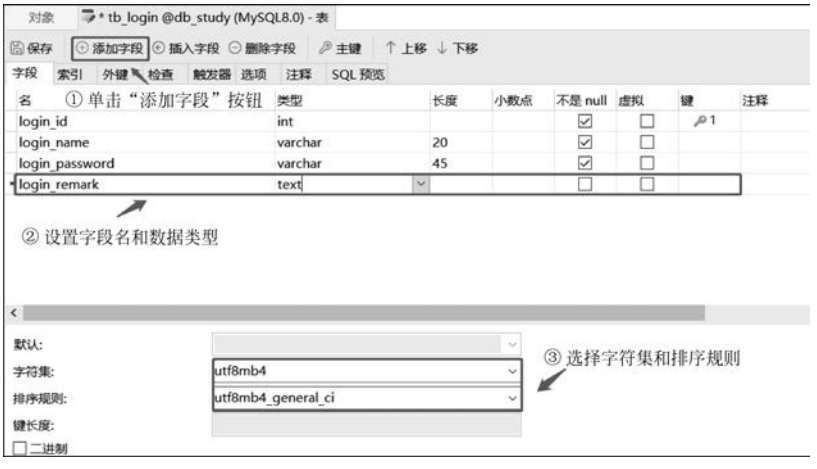

图 5-35　添加 login_remark 字段

(3) 单击“保存”按钮即可,新字段就会添加到 db_study 数据库的 tb_login 数据表中。

常见错误及解决方案

错误 5-1　创建数据表失败

【问题描述】

通过第 4 章的学习,若是创建完 db_study 数据库,直接运行创建数据表语句会报错,如图 5-36 所示。

【解决方案】

错误信息显示“未选择数据库”。用户想要更改信息或操作数据库时,需要先切换到该数据库,才能对其进行修改操作。因为数据表属于数据库,在创建数据表之前,应该先执行“USE 数据库名;”语句指定到数据库 db_study 中进行操作,再创建数据表即可,如图 5-37 所示。

```
mysql> CREATE DATABASE db_study;
Query OK, 1 row affected (0.03 sec)

mysql> CREATE TABLE tb_department
    -> (
    -> department_id CHAR(3),
    -> department_name VARCHAR(50),
    -> department_phone VARCHAR(13),
    -> department_address VARCHAR(50)
    -> );
ERROR 1046 (3D000): No database selected
mysql>
```

图 5-36　创建数据表失败

```
mysql> USE db_study;
Database changed
mysql> CREATE TABLE tb_department
    -> (
    -> department_id CHAR(3),
    -> department_name VARCHAR(50),
    -> department_phone VARCHAR(13),
    -> department_address VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

图 5-37　数据表正确创建方式

错误 5-2　删除数据表失败

【问题描述】

在案例 5-15 中,如果没有创建课程表(tb_course),直接删除 tb_course 数据表会报错,如图 5-38 所示。

【解决方案】

错误信息显示“在 db_study 数据库中没有 tb_course 数据表”,所以想要删除一个数据表,前提是删除该数据库中已创建好的数据表。或者不想出现删除错误,可以在命令中添加 IF EXISTS 参数判断想要删除的数据表是否存在,如果表不存在,则删除数据表的 SQL 语句可以顺利执行,系统不再给出错误提示,但是会发出警告,如图 5-39 所示。

```
mysql> DROP TABLE tb_course;
ERROR 1051 (42S02): Unknown table 'db_study.tb_course'
mysql>
```

图 5-38　数据表删除失败

```
mysql> DROP TABLE IF EXISTS tb_course;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

图 5-39　数据表正确删除方式

扫一扫

自测题

习题

1. 选择题

(1) 下列选项中不是单表约束的是(　　)。

A. 主键约束　　B. 非空约束　　C. 唯一约束　　D. 外键约束

(2) UNIQUE 唯一索引的作用是(　　)。

A. 保证各行在该索引上的值都不重复

B. 保证各行在该索引上的值不为 NULL

C. 保证参加唯一索引的各列，不得再参加其他索引

D. 保证唯一索引不能被删除

(3) MySQL 中的非空约束是(　　)。

A. Foreign Key Constraint　　B. Not Null Constraint

C. Primary Key Constraint　　D. Unique Constraint

(4) 在 SQL 中，创建数据表的命令是(　　)。

A. CREATE DATABASE　　B. CREATE VIEW

C. CREATE TABLE　　D. CREATE INDEX

(5) 查看数据库中所有数据表的命令是(　　)。

A. SHOW DATABASE　　B. SHOW TABLES

C. SHOW DATABASES　　D. SHOW TABLE

2. 填空题

(1) 在 MySQL 中，取值范围最小的整数类型是_____。

(2) 当某字段要使用 AUTO_INCREMENT 属性时，该字段必须是_____类型的数据。

(3) MySQL 数据定义语言中的创建、修改、删除的关键字分别是_____。

(4) VARCHAR 类型长度范围为_____。

(5) SQL 语句中修改表结构的命令是_____。

3. 判断题

(1) 在 MySQL 中不同的数据类型的存储空间不同，取值范围也不同。　　(　　)

(2) MySQL 中 YEAR 类型只有一种存储格式。　　(　　)

(3) 在 MySQL 中，约束是指对表中数据的一种限制。　　(　　)

(4) “ALTER TABLE 旧表名称 RENAME [TO] 新表名称;”语句能对数据表进行重命名。　　(　　)

(5) 在 MySQL 中默认所有类型的值都可以为 NULL。　　(　　)

(6) 在 MySQL 中，使用 DROP TABLE 语句可以删除所有数据表。　　(　　)