第2章 数据库和表的基本操作

学习目标

- ◆ 掌握数据库的基本操作，会对数据库进行增删改查操作
- ◆ 掌握数据表的基本操作，会对数据表进行增删改查操作
- ◆ 了解数据类型，学会 SQL 语句中不同类型数据的表示方式
- ◆ 掌握表的约束，学会使用不同的约束来操作表
- ◆ 掌握索引的作用，会创建和删除索引

在软件开发中，必不可少会使用到数据库和数据表。学会数据库和数据表的基本操作，可以轻松实现数据的管理。本章将针对数据库和数据表的基本操作进行详细地讲解。

2.1 数据库基础知识

2.1.1 创建和查看数据库

MySQL 安装完成后，要想将数据存储到数据库的表中，首先得创建一个数据库。创建数据库就是在数据库系统中划分一块存储数据的空间。在 MySQL 中，创建数据库的基本语法格式如下所示：

```
CREATE DATABASE  数据库名称;
```

在上述语法格式中，“CREATE DATABASE”是固定的 SQL 语句，专门用来创建数据库。“数据库名称”是唯一的，不可重复出现。

【例 2-1】创建一个名称为 itcast 的数据库，SQL 语句如下所示：

```
CREATE DATABASE itcast;
```

执行结果如下所示：

```
mysql> CREATE DATABASE itcast;
Query OK, 1 row affected (0.08 sec)
```

如果看到上述运行结果，说明 SQL 语句执行成功了。为了验证数据库系统中是否创建了名称为 itcast 的数据库，需要查看数据库。在 MySQL 中，查看数据库的 SQL 语句如下所示：

```
SHOW DATABASES;
```

【例 2-2】使用 SHOW 语句查看已经存在的数据库，执行结果如下所示：

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
```

```
| mysql              |
| performance_schema |
| itcast             |
| test               |
+--------------------+
5 rows in set (0.08 sec)
```

从上述执行结果可以看出，数据库系统中存在 5 个数据库。其中，除了在例 2-1 中创建的 itcast 数据库外，其它的数据库都是在 MySQL 安装完成后自动创建的。

创建好数据库之后，要想查看某个已经创建的数据库信息，可以通过 SHOW CREATE DATABASE 语句查看，具体语法格式如下所示：

```
SHOW CREATE DATABASE 数据库名称;
```

【例 2-3】查看创建好的数据库 itcast 的信息，SQL 语句如下所示：

```
SHOW CREATE DATABASE itcast;
```

执行结果如下所示：

```
mysql> SHOW CREATE DATABASE itcast;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| itcast   | CREATE DATABASE `itcast` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

上述执行结果显示出了数据库 itcast 的创建信息，例如，数据库 itcast 的编码方式为 utf8。

2.1.2 修改数据库

MySQL 数据库一旦安装成功，创建的数据库编码也就确定了。但如果想修改数据库的编码，可以使用 ALTER DATABASE 语句实现。修改数据库编码的基本语法格式如下所示：

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

在上述格式中，“数据库名称”指的是要修改的数据库，“编码方式”指的是修改后的数据库编码。

【例 2-4】将数据库 itcast 的编码修改为 gbk，SQL 语句如下所示：

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

为了验证数据库的编码是否修改成功，接下来，使用 SHOW CREATE DATABASE 语句查看修改后的数据库，执行结果如下：

```
mysql> SHOW CREATE DATABASE itcast;
+----------+-----------------------------------------------------------------
------------+
| Database | Create Database
          |
+----------+-----------------------------------------------------------------
------------+
| itcast   | CREATE DATABASE `itcast` /*!40100 DEFAULT CHARACTER SET gbk COLLATE
 gbk_bin */ |
+----------+-----------------------------------------------------------------
```

```
------------+
1 row in set (0.02 sec)
```

从上述执行结果可以看出，数据库 itcast 的编码为 gbk，说明 itcast 数据库的编码信息修改成功了。

2.1.3 删除数据库

删除数据库是将数据库系统中已经存在的数据库删除。成功删除数据库后，数据库中的所有数据都将被清除，原来分配的空间也将被回收。在 MySQL 中，删除数据库的基本语法格式如下所示：

```
DROP DATABASE 数据库名称;
```

在上述语法格式中，“DROP DATABASE”是删除数据库的 SQL 语句，“数据库名称”是要删除的数据库名称。需要注意的是，如果要删除的数据库不存在，则删除会失败。

【例 2-5】删除名称为 itcast 的数据库，SQL 语句如下所示：

```
DROP DATABASE itcast;
```

为了验证删除数据库的操作是否成功，接下来，使用 SHOW DATABASES 语句查看已经存在的数据库，执行结果如下所示：

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.05 sec)
```

从上述执行结果可以看出，数据库系统中已经不存在名称为 itcast 的数据库了，说明 itcast 数据库被成功删除了。

2.2 数据类型

使用 MySQL 数据库存储数据时，不同的数据类型决定了 MySQL 存储数据方式的不同。为此，MySQL 数据库提供了多种数据类型，其中包括整数类型、浮点数类型、定点数类型、日期和时间类型、字符串类型和二进制类型。接下来，本节将针对这些数据类型进行详细地讲解。

2.2.1 整数类型

在 MySQL 数据库中，经常需要存储整数数值。根据数值取值范围的不同，MySQL 中的整数类型可分为 5 种，分别是 TINYINT、SMALLINT、MEDIUMINT、INT 和 BIGINT。表 2-1 列举了 MySQL 不同整数类型所对应的字节大小和取值范围。

表2-1　MySQL 整数类型

数据类型	字节数	无符号数的取值范围	有符号数的取值范围
TINYINT	1	0~255	-128~127

SMALLINT	2	0~65535	-32768~32768
MEDIUMINT	3	0~16777215	-8388608~8388608
INT	4	0~4294967295	-2147483648~2147483648
BIGINT	8	0~18446744073709551615	-9223372036854775808~9223372036854775808

从表 2-1 中可以看出，不同整数类型所占用的字节数和取值范围都是不同的。其中，占用字节数最小的是 TINYINT，占用字节数最大的是 BIGINT。需要注意的是，不同整数类型的取值范围可以根据字节数计算出来，例如，TINYINT 类型的整数占用 1 个字节，1 个字节是 8 位，那么，TINYINT 类型无符号数的最大值就是 2^8-1，即 255，TINYINT 类型有符号数的最大值就是 2^7-1，即 127。同理可以算出其它不同整数类型的取值范围。

2.2.2 浮点数类型和定点数类型

在 MySQL 数据库中，存储的小数都是使用浮点数和定点数来表示的。浮点数的类型有两种，分别是单精度浮点数类型（FLOAT）和双精度浮点类型（DOUBLE）。而定点数类型只有 DECIMAL 类型。表 2-2 列举了 MySQL 中浮点数和定点数类型所对应的字节大小及其取值范围。

表2-2 MySQL 浮点数和定点数类型

数据类型	字节数	有符号的取值范围	无符号的取值范围
FLOAT	4	-3.402823466E+38~ -1.175494351E-38	0 和 1.175494351E-38~ 3.402823466E+38
DOUBLE	4	-1.7976931348623157E+308~ 2.2250738585072014E-308	0 和 2.2250738585072014E-308~ 1.7976931348623157E+308
DECIMAL(M，D)	M+2	-1.7976931348623157E+308~ 2.2250738585072014E-308	0 和 2.2250738585072014E-308~ 1.7976931348623157E+308

从表 2-2 中可以看出，DECIMAL 类型的取值范围与 DOUBLE 类型相同。需要注意的是，DECIMAL 类型的有效取值范围是由 M 和 D 决定的，其中，M 表示的是数据的长度，D 表示的是小数点后的长度。比如，将数据类型为 DECIMAL(6，2) 的数据 3.1415 插入数据库后，显示的结果为 3.14。

2.2.3 日期与时间类型

为了方便在数据库中存储日期和时间，MySQL 提供了表示日期和时间的数据类型，分别是 YEAR、DATE、TIME、DATETIME 和 TIMESTAMP。表 2-3 列举了这些 MySQL 中日期和时间数据类型所对应的字节数、取值范围、日期格式以及零值。

表2-3 MySQL 日期和时间类型

数据类型	字节数	取值范围	日期格式	零值
YEAR	1	1901~2155	YYYY	0000
DATE	4	1000-01-01~9999-12-3	YYYY-MM-DD	0000-00-00
TIME	3	-838:59:59~838:59:59	HH:MM:SS	00:00:00
DATETIME	8	1000-01-01 00:00:00~ 9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	0000-00-00 00:00:00
TIMESTAMP	4	1970-01-01 00:00:01~ 2038-01-19 03:14:07	YYYY-MM-DD HH:MM:SS	0000-00-00 00:00:00

从表 2-3 中可以看出，每种日期和时间类型的取值范围都是不同的。需要注意的是，如果插入的数值不合法，系统会自动将对应的零值插入数据库中。

为了大家更好地学习日期和时间类型，接下来，将表 2-3 中的类型进行详细讲解，具体如下：

1、YEAR 类型

YEAR 类型用于表示年份，在 MySQL 中，可以使用以下三种格式指定 YEAR 类型的值：

（1）使用 4 位字符串或数字表示，范围为‘1901’~‘2155’或 1901~2155。

例如，输入‘2014’或 2014，插入到数据库的值均为 2014。

（2）使用 2 位字符串表示，范围为‘00’~‘99’，其中，‘00’~‘69’范围的值会被转换为 2000~2069 范围的 YEAR 值，‘70’~‘99’范围的值会被转换为 1970~1999 范围的 YEAR 值。

例如，输入‘14’，插入到数据库的值为 2014。

（3）使用 2 位数字表示，范围为 1~99，其中，1~69 范围的值会被转换为 2001~2069 范围的 YEAR 值，70~99 范围的值会被转换为 1970~1999 范围的 YEAR 值。例如，输入 14，插入到数据库的值为 2014。

需要注意的是，当使用 YEAR 类型时，一定要区分‘0’和 0。因为字符串格式的‘0’表示的是 YEAR 值是 2000，而数字格式的 0 表示的 YEAR 值是 0000。

2、DATE 类型

DATE 类型用于表示日期值，不包含时间部分。在 MySQL 中，可以使用以下四种格式指定 DATE 类型的值：

（1）以‘YYYY-MM-DD’或者‘YYYYMMDD’字符串格式表示。

例如，输入‘2014-01-21’或‘20140121’，插入数据库的日期都为 2014-01-21。

（2）以‘YY-MM-DD’或者‘YYMMDD’字符串格式表示。YY 表示的是年，范围为‘00’~‘99’，其中‘00’~‘69’范围的值会被转换为 2000~2069 范围的值，‘70’~‘99’范围的值会被转换为 1970~1999 范围的值。

例如，输入‘14-01-21’或‘140121’，插入数据库的日期都为 2014-01-21。

（3）以 YY-MM-DD 或者 YYMMDD 数字格式表示。

例如，输入 14-01-21 或 140121，插入数据库的日期都为 2014-01-21。

（4）使用 CURRENT_DATE 或者 NOW()表示当前系统日期。

3、TIME 类型

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。在 MySQL 中，可以使用以下三种格式指定 TIME 类型的值：

（1）以‘D HH:MM:SS’字符串格式表示。其中，D 表示日，可以取 0~34 之间的值，插入数据时，小时的值等于（D*24+HH）。

例如，输入‘2 11:30:50’，插入数据库的日期为 59:30:50。

（2）以‘HHMMSS’字符串格式或者 HHMMSS 数字格式表示。

例如，输入‘345454’或 345454，插入数据库的日期为 34:54:54。

（3）使用 CURRENT_TIME 或 NOW()输入当前系统时间。

4、DATETIME 类型

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。在 MySQL 中，可以使用以下四种格式指定 DATETIME 类型的值：

（1）以‘YYYY-MM-DD HH:MM:SS’或者‘YYYYMMDDHHMMSS’字符串格式表示的日期和时间，取值范围为‘1000-01-01 00:00:00’~‘9999-12-3 23:59:59’。

例如，输入‘2014-01-22 09:01:23’或 20140122090123，插入数据库的 DATETIME 值都为 2014-01-22 09:01:23。

（2）以‘YY-MM-DD HH:MM:SS’或者‘YYMMDDHHMMSS’字符串格式表示的日期和时间，其中 YY 表示年，取值范围为‘00’~‘99’。与 DATE 类型中的 YY 相同，‘00’~‘69’范围的值会被转换为 2000~2069 范围的值，‘70’~‘99’范围的值会被转换为 1970~1999 范围的值。

（3）以 YYYYMMDDHHMMSS 或者 YYMMDDHHMMSS 数字格式表示的日期和时间。

例如，插入 20140122090123 或者 140122090123，插入数据库的 DATETIME 值都为 2014-01-22 09:01:23。

（4）使用 NOW 来输入当前系统的日期和时间。

5、TIMESTAMP 类型

TIMESTAMP 类型用于表示日期和时间，它的显示形式同 DATETIME 相同，但取值范围比 DATETIME 小。下面介绍几种 TIMESTAMP 类型与 DATATIME 类型不同的形式，具体如下：

（1）使用 CURRENT_TIMESTAMP 来输入系统当前日期和时间。

（2）输入 NULL 时，系统会输入系统当前日期和时间。

（3）无任何输入时，系统会输入系统当前日期和时间。

2.2.4 字符串和二进制类型

为了存储字符串、图片和声音等数据，MySQL 提供了字符串和二进制类型。表 2-4 列举了 MySQL 中的字符串和二进制类型。

表2-4 MySQL 字符串和二进制类型

数据类型	类型说明
CHAR	用于表示固定长度的字符串
VARCHAR	用于表示可变长度的字符串
BINARY	用于表示固定长度的二进制数据
VARBINARY	用于表示可变长度的二进制数据
BOLB	用于表示二进制大数据
TEXT	用于表示大文本数据
ENUM	表示枚举类型，只能存储一个枚举字符串值
SET	表示字符串对象，可以有零或多个值
BIT	表示位字段类型

表 2-4 列举的字符串和二进制类型中，不同数据类型具有不同的特点，接下来，针对这些数据类型进行详细地讲解，具体如下：

1、 CHAR 和 VARCHAR 类型

CHAR 和 VARCHAR 类型都用来表示字符串数据，不同的是，VARCHAR 可以存储可变长度的字符串。在 MySQL 中，定义 CHAR 和 VARCHAR 类型的方式如下所示：

```
CHAR(M) 或 VARCHAR(M)
```

在上述定义方式中，M 指的是字符串的最大长度。为了帮助大家更好地理解 CHAR 和 VARCHAR 之间的区别，接下来，以 CHAR(4)和 VARCHAR(4)为例进行说明，具体如表 2-5 所示。

表2-5 CHAR(4)和 VARCHAR(4)对比

插入值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
‘’	‘’	4 个字节	‘’	1 个字节
‘ab’	‘ab’	4 个字节	‘ab’	3 个字节

‘abc’	‘abc’	4 个字节	‘abc’	4 个字节
‘abcd’	‘abcd’	4 个字节	‘abcd’	5 个字节
‘abcdef’	‘abcd’	4 个字节	‘abcd’	5 个字节

从表 2-5 中可以看出，当数据为 CHAR(4)类型时，不管插入值的长度是多少，所占用的存储空间都是 4 个字节。而 VARCHAR(4)所对应的数据所占用的字节数为实际长度加 1。

2、BINARY 和 VARBINARY 类型

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR，不同的是，它们所表示的是二进制数据。定义 BINARY 和 VARBINARY 类型的方式如下所示：

```
BINARY(M) 或 VARBINARY(M)
```

在上述格式中，M 指的是二进制数据的最大字节长度。

需要注意的是，BINARY 类型的长度是固定的，如果数据的长度不足最大长度，将在数据的后面用“\0”补齐，最终达到指定长度。例如，指定数据类型为 BINARY(3)，当插入 a 时，实际存储的数据为“a\0\0”，当插入 ab 时，实际存储的数据为“ab\0”。

3、TEXT 类型

TEXT 类型用于表示大文本数据，例如，文章内容、评论等，它的类型分为四种，具体如表 2-6 所示。

表2-6　TEXT 类型

数据类型	存储范围
TINYTEXT	0~255 字节
TEXT	0~65535 字节
MEDIUMTEXT	0~16777215 字节
LONGTEXT	0~4294967295 字节

4、BLOB 类型

BLOB 类型是一种特殊的二进制类型，它用于表示数据量很大的二进制数据，例如图片、PDF 文档等。BLOB 类型分为四种，具体如表 2-7 所示。

表2-7　BLOB 类型

数据类型	存储范围
TINYBLOB	0~255 字节
BLOB	0~65535 字节
MEDIUMBLOB	0~16777215 字节
LONGBLOB	0~4294967295 字节

需要注意的是，BLOB 类型与 TEXT 类型很相似，但 BLOB 类型数据是根据二进制编码进行比较和排序，而 TEXT 类型数据是根据文本模式进行比较和排序。

5、ENUM 类型

ENUM 类型又称为枚举类型，定义 ENUM 类型的数据格式如下所示：

```
ENUM('值 1', '值 2', '值 3'……'值 n')
```

在上述格式中，('值 1', '值 2', '值 3'……'值 n')称为枚举列表，ENUM 类型的数据只能从枚举列表中取，并且只能取一个。需要注意的是，枚举列举中的每个值都有一个顺序编号，MySQL 中存入的就是这个顺序编号，而不是列表中的值。

6、SET 类型

SET 类型用于表示字符串对象，它的值可以有零个或多个，SET 类型数据的定义格式与 ENUM 类型类似，具体语法格式如下所示：

```
SET('值 1', '值 2', '值 3'……'值 n')
```

与 ENUM 类型相同，('值 1', '值 2', '值 3'……'值 n')列表中的每个值都有一个顺序编号，MySQL 中存入的也是这个顺序编号，而不是列表中的值。

7、BIT 类型

BIT 类型用于表示二进制数据。定义 BIT 类型的基本语法格式如下所示：

```
BIT(M)
```

在上述格式中，M 用于表示每个值的位数，范围为 1~64。需要注意的是，如果分配的 BIT(M)类型的数据长度小于 M，将在数据的左边用 0 补齐。例如，为 BIT(6)分配值 b’101’的效果与分配 b’000101’相同。

2.3 数据表的基本操作

2.3.1 创建数据表

数据库创建成功后，就需要创建数据表。所谓创建数据表指的是在已存在的数据库中建立新表。需要注意的是，在操作数据表之前，应该使用“USE 数据库名”指定操作是在哪个数据库中进行，否则会抛出“No database selected”错误。创建数据表的基本语法格式如下所示：

```
CREATE TABLE 表名
(
  字段名 1,数据类型[完整性约束条件],
  字段名 2,数据类型[完整性约束条件],
  . . . . . .
  字段名 n,数据类型[完整性约束条件],
)
```

在上述语法格式中，“表名”指的是创建的数据表名称，“字段名”指的是数据表的列名，“完整性约束条件”指的是字段的某些特殊约束条件，关于表的约束，将在下个小节进行详细讲解。

【例 2-6】创建一个用于存储学生成绩的表 tb_grade，如表 2-8 所示。

表2-8 tb_grade 表

字段名称	数据类型	备注说明
id	INT(11)	学生的编号
name	VARCHAR(20)	学生的姓名
grade	FLOAT	学生的成绩

要想创建表 2-8 所示的数据表，需要首先创建一个数据库，SQL 语句如下：

```
CREATE DATABASE itcast;
```

选择创建表的数据库，SQL 语句如下：

```
USE itcast;
```

创建数据表的 SQL 语句如下所示：

```
CREATE TABLE tb_grade
(
   id      INT(11),
   name   VARCHAR(20),
   grade   FLOAT
);
```

为了验证数据表是否创建成功，需要使用 SHOW TABLES 语句进行查看，具体执行结果如下所示：

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_itcast |
+------------------+
| tb_grade         |
+------------------+
1 row in set (0.03 sec)
```

从上述执行结果可以看出，itcast 数据库中已经存在了数据表 tb_grade，说明数据表创建成功了。

2.3.2 查看数据表

使用 SQL 语句创建好数据表后，可以通过查看数据表结构的定义，以确认数据表的定义是否正确。在 MySQL 中，查看数据表的方式有两种，具体如下：

1、使用 SHOW CREATE TABLE 查看数据表

在 MySQL 中，SHOW CREATE TABLE 语句不仅可以查看创建表时的定义语句，还可以查看表的字符编码。SHOW CREATE TABLE 语句的基本语法格式如下所示：

```
SHOW CREATE TABLE 表名;
```

在上述格式中，“表名”指的是要查询数据表的名称。

【例 2-7】使用 SHOW CREATE TABLE 语句查看 tb_grade 表，SQL 语句如下所示：

```
SHOW CREATE TABLE tb_grade;
```

执行结果如下所示：

```
mysql> SHOW CREATE TABLE tb_grade;
+----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-----------------------------------------+
| Table    | Create Table

                                        |
+----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-----------------------------------------+
| tb_grade | CREATE TABLE 'tb_grade' (
  'id' int(11) DEFAULT NULL,
  'name' varchar(20) COLLATE utf8_bin DEFAULT NULL,
  'grade' float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+----------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-----------------------------------------+
1 row in set (0.06 sec)
```

从上述执行结果可以看出，tb_grade 数据表的定义信息显示了出来，但是显示的结果非常混乱，这时，我们可以在 SHOW CREATE TABLE 语句的表名之后加上参数“\G”，使显示结果整齐美观，具体执行结果如下所示：

```
mysql> SHOW CREATE TABLE tb_grade\G
*************************** 1. row ***************************
       Table: tb_grade
Create Table: CREATE TABLE 'tb_grade' (
  'id' int(11) DEFAULT NULL,
  'name' varchar(20) COLLATE utf8_bin DEFAULT NULL,
  'grade' float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

2、使用 DESCRIBE 语句查看数据表

在 MySQL 中，使用 DESCRIBE 语句可以查看表的字段信息，其中包括字段名、字段类型等信息。DESCRIBE 语句的基本语法格式如下所示：

```
DESCRIBE 表名;
```

或简写为：

```
DESC 表名;
```

【例 2-8】使用 DESCRIBE 语句查看 tb_grade 表，SQL 语句如下所示：

```
DESCRIBE tb_grade;
```

执行结果如下所示：

```
mysql> DESCRIBE tb_grade;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| grade | float       | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.06 sec)
```

上述执行结果显示出了 tb_grade 表的字段信息，接下来，针对执行结果中的不同字段进行详细讲解，具体如下：

- NULL：表示该列是否可以存储 NULL 值。
- Key：表示该列是否已经编制索引。
- Default：表示该列是否有默认值。
- Extra：表示获取到的与给定列相关的附加信息。

2.3.3 修改数据表

有时候，希望对表中的某些信息进行修改，这时就需要修改数据表。所谓修改数据表指的是修改数据库中已经存在的数据表结构，比如，修改表名、修改字段名、修改字段的数据类型等等。在 MySQL 中，修改数据表的操作都是使用 ALTER TABLE 语句，接下来，本节将针对修改数据表的相关操作进行详细地讲解，具体如下：

1、 修改表名

在数据库中，不同的数据表是通过表名来区分的。在 MySQL 中，修改表名的基本语法格式如下所示：

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

在上述格式中，“旧表名”指的是修改前的表名，“新表名”指的是修改后的表名，关键字 TO 是可选的，其在 SQL 语句中是否出现不会影响语句的执行。

【例 2-9】将数据库 itcast 中的 tb_grade 表名改为 grade 表。

在修改数据库表名之前，首先使用 SHOW TABLES 语句查看数据库中的所有表，执行结果如下：

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_itcast |
+-------------------+
| tb_grade          |
+-------------------+
1 row in set (0.00 sec)
```

上述语句执行完毕后，使用 ALTER TABLE 将表名 tb_grade 修改为 grade，SQL 语句如下：

```
ALTER TABLE tb_grade RENAME TO grade;
```

为了检测表名是否修改正确，再次使用 SHOW TABLES 语句查看数据库中的所有表，执行结果如下所示：

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_itcast  |
+-------------------+
| grade             |
+-------------------+
1 row in set (0.00 sec)
```

从上述执行结果可以看出，数据库中的 tb_grade 表名成功被修改为 grade 了。

2、 修改字段名

数据表中的字段是通过字段名来区分的。在 MySQL 中，修改字段名的基本语法格式如下所示：

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

在上述格式中，“旧字段名”指的是修改前的字段名，“新字段名”指的是修改后的字段名，“新数据类型”指的是修改后的数据类型。需要注意的是，新数据类型不能为空，即使新字段与旧字段的数据类型相同，也必须将新数据类型设置为与原来一样的数据类型。

【例 2-10】将数据表 grade 中的 name 字段改为 username，数据类型保持不变，SQL 语句如下所示：

```
ALTER TABLE grade CHANGE name username VARCHAR(20);
```

为了验证字段名是否修改成功，通过 DECS 语句查看 grade 表的结构，执行结果如下所示：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

```
3 rows in set (0.17 sec)
```

从上述执行结果可以看出，数据表 grade 中的字段名 name 成功修改成了 username。

3、 修改字段的数据类型

修改字段的数据类型，就是将字段的数据类型转为另外一种数据类型。在 MySQL 中修改字段数据类型的基本语法格式如下所示：

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

在上述格式中，“表名”指的是要修改字段所在的表名，“字段名”指的是要修改的字段，“数据类型”指的是修改后的字段的数据类型。

【例 2-11】将数据表 grade 中的 id 字段的数据类型由 INT(11)修改为 INT(20)。

在执行修改字段的数据类型之前，首先使用 DESC 查看 grade 表的结构，如下所示：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

从上述执行结果可以看出，id 字段的数据类型为 INT(11)。接下来，使用 ALTER 语句修改 id 字段的数据类型，SQL 语句如下所示：

```
ALTER TABLE grade MODIFY id INT(20);
```

为了验证 id 字段的数据类型是否修改成功，再次使用 DECS 查看 grade 数据表，执行结果如下：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(20)     | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

从上述结果可以看出，grade 表中的 id 字段的数据类型成功被修改成了 INT(20)。

4、 添加字段

在创建数据表时，表中的字段就已经定义好了。但是，如果想在创建好的数据表中添加字段，则需要通过 ALTER TABLE 语句进行增加。在 MySQL 中，添加字段的基本语法格式如下所示：

```
ALTER TABLE 表名 ADD  新字段名  数据类型
     [约束条件][FIRST|AFTER 已存在字段名]
```

在上述格式中，“新字段名”为添加字段的名称，“FIRST”为可选参数，用于将新添加的字段设置为表的第一个字段，“AFTER”也为可选参数，用于将新添加的字段添加到指定的“已存在字段名”的后面。

【例 2-12】在数据表 grade 中添加一个没有约束条件的 INT 类型的字段 age，SQL 语句如下：

```
ALTER TABLE grade ADD age INT(10);
```

为了验证字段 age 是否添加成功，接下来，使用 DESC 语句查看数据表 grade，执行结果如下：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(20)     | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
| age      | int(10)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.11 sec)
```

从上述执行结果可以看出，grade 表中添加了一个 age 字段，并且字段的数据类型为 INT(10)。

5、删除字段

数据表创建成功后，不仅可以修改字段，还可以删除字段。所谓删除字段指的是将某个字段从表中删除。在 MySQL 中，删除字段的基本语法格式如下所示：

```
ALTER TABLE 表名 DROP 字段名;
```

在上述格式中，“字段名”指的是要删除的字段的名称。

【例 2-13】删除 grade 表中的 age 字段，SQL 语句如下：

```
ALTER TABLE grade DROP age;
```

为了验证 age 字段是否删除，接下来，使用 DESC 语句查看 grade 表，执行结果如下：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(20)     | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| grade    | float       | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

从上述执行结果可以看出，grade 表中已经不存在 age 字段，说明 age 字段被成功删除了。

6、修改字段的排列位置

创建数据表的数据，字段在表中的位置已经确定了。但要修改字段在表中的排列位置，则需要使用 ALTER TABLE 语句来处理。在 MySQL 中，修改字段排列位置的基本语法格式如下：

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

在上述格式中，“字段名 1”指的是修改位置的字段，“数据类型”指的是字段 1 的数据类型，“FIRST”为可选参数，指的是将字段 1 修改为表的第一个字段，“AFTER 字段名 2”是将字段 1 插入到字段 2 的后面。

【例 2-14】将数据表 grade 的 username 字段修改为表的第一个字段，执行的 SQL 语句如下：

```
ALTER TABLE grade MODIFY username VARCHAR(20) FIRST;
```

为了验证 username 字段是否修改为表的第一个字段，接下来，使用 DESC 语句查看数据表，执行结果如下：

```
mysql> DESC grade;
+----------+-------------+------+-----+---------+-------+
```

```
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| username | varchar(20)  | YES  |     | NULL    |       |
| id       | int(20)      | YES  |     | NULL    |       |
| grade    | float        | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

从上述执行结果可以看出，username 字段为表的第一个字段，说明 username 字段的排列位置被成功修改了。

【例 2-15】将数据表 grade 的 id 字段插入 grade 字段后面，执行的 SQL 语句如下：

```
ALTER TABLE grade MODIFY id INT(20) AFTER grade;
```

为了验证 id 字段是否插入到 grade 字段后面，接下来，使用 DESC 语句查看数据表，执行结果如下：

```
mysql> DESC grade;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| username | varchar(20)  | YES  |     | NULL    |       |
| grade    | float        | YES  |     | NULL    |       |
| id       | int(20)      | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

从上述结果可以看出，id 字段位于 grade 字段后面，说明 id 字段的排列位置被成功修改了。

2.3.4 删除数据表

删除数据表是指删除数据库中已存在的表。在删除数据表的同时，数据表中存储的数据都将被删除。需要注意的是，创建数据表时，表和表之间可能会存在关联，要删除这些被其它表关联的表比较复杂，将在后面的章节进行讲解。本节讲解的是删除没有关联关系的数据表。

在 MySQL 中，直接使用 DROP TABLE 语句就可以删除没有被其它表关联的数据表，其基本的语法格式如下所示：

```
DROP TABLE 表名;
```

在上述格式中，“表名”指的是要删除的数据表。

【例 2-16】删除数据表 grade，SQL 语句如下：

```
DROP TABLE grade;
```

为了验证数据表 grade 是否被删除成功，使用 DESC 语句查看数据表，执行结果如下：

```
mysql> DESC grade;
ERROR 1146 (42S02): Table 'itcast.grade' doesn't exist
```

从上述结果可以看出，grade 表已经不存在了，说明数据表 grade 被成功删除了。

2.4 表的约束

为了防止数据表中插入错误的数据，在 MySQL 中，定义了一些维护数据库完整性的规则，即表的约束。表 2-9 列举了常见的表的约束。

表2-9 表的约束

约束条件	说明
PRIMARY KEY	主键约束，用于唯一标识对应的记录
FOREIGN KEY	外键约束
NOT NULL	非空约束
UNIQUE	唯一性约束
DEFAULT	默认值约束，用于设置字段的默认值

表 2-9 列举的约束条件都是针对表中字段进行限制，从而保证数据表中数据的正确性和唯一性。由于 FOREIGN KEY 约束条件涉及到多表操作，因此，本节只针对除 FOREIGN KEY 外的其它约束进行详细地讲解。

2.4.1 主键约束

在 MySQL 中，为了快速查找表中的某条信息，可以通过设置主键来实现。主键约束是通过 PRIMARY KEY 定义的，它可以唯一标识表中的记录，这就好比身份证可以用来标识人的身份一样。在 MySQL 中，主键约束分为两种，具体如下：

1、单字段主键

单字段主键指的是由一个字段构成的主键，其基本的语法格式如下所示：

```
字段名 数据类型 PRIMARY KEY
```

【例 2-17】创建一个数据表 example01，并设置 id 作为主键，SQL 语句如下：

```
CREATE TABLE example01( id INT PRIMARY KEY,
                        name VARCHAR(20),
                           grade FLOAT);
```

上述 SQL 语句执行后，example01 表中创建了 id、name 和 grade 三个字段，其中，id 字段是主键。

2、多字段主键

多字段主键指的是多个字段组合而成的主键，其基本的语法格式如下所示：

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

在上述格式中，“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 2-18】创建一个数据表 example02，在表中将 stu_id 和 course_id 两个字段共同作为主键，SQL 语句如下：

```
CREATE TABLE example02( stu_id INT,
                        course_id INT,
                        grade FLOAT,
                        PRIMARY KEY(stu_id,course_id)
                        );
```

上述 SQL 语句执行后，example02 表中包含了 stu_id、course_id 和 grade 三个字段，其中，stu_id 和 course_id 两个字段组合可以唯一确定一条记录。

注意：

每个数据表中最多只能有一个主键约束，定义为 PRIMARY KEY 的字段不能有重复值且不能为 NULL 值。

2.4.2 非空约束

非空约束指的是字段的值不能为 NULL，在 MySQL 中，非空约束是通过 NOT NULL 定义的，其基本的语法格式如下所示：

```
字段名 数据类型 NOT NULL;
```

【例 2-19】创建一个数据表 example04，将表中的 name 字段设置为非空约束，SQL 语句如下：

```
CREATE TABLE example04( id INT PRIMARY KEY,
                        name VARCHAR(20) NOT NULL,
                        grade FLOAT);
```

上述 SQL 语句执行后，example04 表中包含了 id、name 和 grade 三个字段。其中，id 字段为主键， name 字段为非空字段。需要注意的是，在同一个数据表中可以定义多个非空字段。

2.4.3 唯一约束

唯一约束用于保证数据表中字段的唯一性，即表中字段的值不能重复出现。唯一约束是通过 UNIQUE 定义的，其基本的语法格式如下所示：

```
字段名 数据类型 UNIQUE;
```

【例 2-20】创建一个数据表 example05，将表中的 stu_id 设置为唯一约束，SQL 语句如下：

```
CREATE TABLE example05( id INT PRIMARY KEY,
                        stu_id INT UNIQUE,
                        name VARCHAR(20) NOT NULL
                       );
```

上述 SQL 语句执行后，example05 表中包含了 id、stu_id 和 name 三个字段。其中，id 字段为主键，stu_id 字段为唯一值，该字段的值不能重复，name 字段的值不能为空值。

2.4.4 默认约束

默认约束用于给数据表中的字段指定默认值，即当在表中插入一条新记录时，如果没有给这个字段赋值，那么，数据库系统会自动为这个字段插入默认值。默认值是通过 DEFAULT 关键字定义的，其基本的语法格式如下所示：

```
字段名 数据类型  DEFAULT 默认值;
```

【例 2-21】创建一个数据表 example06，将表中的 grade 字段的默认值设置为 0，SQL 语句如下：

```
CREATE TABLE example06( id INT PRIMARY KEY AUTO_INCREMENT,
                        stu_id INT UNIQUE,
                        grade FLOAT DEFAULT 0
                       );
```

上述 SQL 语句执行后，example06 表中包含 id、stu_id 和 grade 三个字段。其中，id 字段为主键，stu_id 字段的值唯一，grade 字段的默认值为 0。

2.5 设置表的字段值自动增加

在数据表中，若想为表中插入的新记录自动生成唯一的 ID，可以使用 AUTO_INCREMENT 约束来实现。AUTO_INCREMENT 约束的字段可以是任何整数类型。默认情况下，该字段的值是从 1 开始自增的。使用 AUTO_INCREMENT 设置表字段值自动增加的基本语法格式如下所示：

```
字段名 数据类型 AUTO_INCREMENT;
```

【例 2-22】创建一个数据表 example05，将表中的 id 字段设置为自动增加，SQL 语句如下：

```
CREATE TABLE example05( id INT PRIMARY KEY AUTO_INCREMENT,
                        stu_id INT UNIQUE,
                        grade FLOAT
                       );
```

上述 SQL 语句执行后，example05 表中 包含 3 个字段。其中，id 字段为主键，且每插入一条新记录，id 的值会自动增加，stu_id 字段的值唯一，grade 的值为 FLOAT 类型。

2.6 索引

在数据库操作中，经常需要查找特定的数据，例如，当执行“select * from student where id=10000”语句时，MySQL 数据库必须从第 1 条记录开始遍历，直到找到 id 为 10000 的数据，这样的效率显然非常低。为此，MySQL 允许建立索引来加快数据表的查询和排序。接下来，本节将针对数据库的索引进行详细讲解。

2.6.1 索引的概念

数据库的索引好比新华字典的音序表，它是对数据库表中一列或多列的值进行排序后的一种结构，其作用就是提高表中数据的查询速度。MySQL 中的索引分为很多种，具体如下：

1、普通索引

普遍索引是由 KEY 或 INDEX 定义的索引，它是 MySQL 中的基本索引类型，可以创建在任何数据类型中，其值是否唯一和非空由字段本身的约束条件所决定。例如，在 grade 表的 stu_id 字段上建立一个普通索引，查询记录时，就可以根据该索引进行查询了。

2、唯一性索引

唯一性索引是由 UNIQUE 定义的索引，该索引所在字段的值必须是唯一的。例如，在 grade 表的 id 字段上建立唯一性索引，那么，id 字段的值就必须是唯一的。

3、全文索引

全文索引是由 FULLTEXT 定义的索引，它只能创建在 CHAR、VARCHAR 或 TEXT 类型的字段上，而且，现在只有 MyISAM 存储引擎支持全文索引。

4、单列索引

单列索引指的是在表中单个字段上创建索引，它可以是普通索引、唯一索引或者全文索引，只要保证该索引只对应表中一个字段即可。

5、多列索引

多列索引指的是在表中多个字段上创建索引，只有在查询条件中使用了这些字段中的第一个字段时，该索引才会被使用。例如，在 grade 表的 id、name 和 score 字段上创建一个多列索引，那么，只有查询条件中使用了 id 字段时，该索引才会被使用。

6、空间索引

空间索引是由 SPATIAL 定义的索引，它只能创建在空间数据类型的字段上。MySQL 中的空间数据类型有 4 种，分别是 GEOMETRY、POINT、LINESTRING 和 POLYGON。需要注意的是，创建空间索引的字段，必须将其声明为 NOT NULL，并且空间索引只能在存储引擎为 MyISAM 的表中创建。

需要注意的是，虽然索引可以提高数据的查询速度，但索引会占用一定的磁盘空间，并且在创建和维护索引时，其消耗的时间是随着数据量的增加而增加的。因此，使用索引时，应该综合考虑索引的优点和缺点。

2.6.2 创建索引

要想使用索引提高数据表的访问速度，首先得创建一个索引。创建索引的方式有三种，具体如下：

一、创建表的时候创建索引

创建表的时候可以直接创建索引，这种方式最简单、方便，其基本的语法格式如下所示：

```
CREATE TABLE 表名（字段名 数据类型[完整性约束条件],
                  字段名 数据类型[完整性约束条件],
                  . . . . . .
                      字段名 数据类型
                    [UNIQUE|FULLTEXT|SPATIAL] INDEX|KEY
                       [别名] (字段名 1 [(长度)]) [ASC|DESC])
                    );
```

关于上述语法的相关解释具体如下：

- UNIQUE：可选参数，表示唯一性约束
- FULLTEXT：可选参数，表示全文约束
- SPATIAL：可选参数，表示空间约束
- INDEX 和 KEY：用来表示字段的索引，二者选一即可
- 别名：可选参数，表示创建的索引的名称
- 字段名 1：指定索引对应字段的名称
- 长度：可选参数，用于表示索引的长度
- ASC 和 DESC：可选参数，其中，ASC 表示升序排列，DESC 表示降序排列

为了帮助大家更好地了解如何在创建表的时候创建索引，接下来，通过具体的案例，分别对 MySQL 中的 6 种索引类型进行讲解，具体如下：

1、创建普通索引

【例 2-23】在 t1 表中 id 字段上建立索引，SQL 语句如下：

```
CREATE TABLE t1(id INT,
                name VARCHAR(20),
                 score FLOAT,
                 INDEX (id)
                );
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql>  SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE 't1' (
  'id' int(11) DEFAULT NULL,
  'name' varchar(20) COLLATE utf8_bin DEFAULT NULL,
  'score' float DEFAULT NULL,
  KEY 'id' ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，id 字段上已经创建了一个名称为 id 的索引。为了查看索引是否被使用，可以使用 EXPLAIN 语句进行查看，SQL 代码如下：

```
EXPLAIN SELECT * FROM t1 WHERE id=1 \G
```

执行结果如下所示：

```
mysql> EXPLAIN SELECT * FROM t1 WHERE id=1 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ref
possible_keys: id
          key: id
      key_len: 5
          ref: const
         rows: 1
        Extra: Using where
1 row in set (0.03 sec)
```

从上述执行结果可以看出，possible_keys 和 key 的值都为 id，说明 id 索引已经存在，并且已经开始被使用了。

2、创建唯一性索引

【例 2-24】创建一个表名为 t2 的表，在表中的 id 字段上建立索引名为 unique_id 的唯一性索引，并且按照升序排列，SQL 语句如下：

```
CREATE TABLE t2(id INT NOT NULL,
                name VARCHAR(20) NOT NULL,
                score FLOAT,
                UNIQUE INDEX unique_id(id ASC)
                );
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE 't2' (
  'id' int(11) NOT NULL,
```

```
  'name' varchar(20) COLLATE utf8_bin NOT NULL,
  'score' float DEFAULT NULL,
  UNIQUE KEY 'unique_id' ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，id 字段上已经建立了一个名称为 unique_id 的唯一性索引。

3、创建全文索引

【例 2-25】创建一个表名为 t3 的表，在表中的 name 字段上建立索引名为 fulltext_name 的全文索引，SQL 语句如下：

```
CREATE TABLE t3(id INT NOT NULL,
                name VARCHAR(20) NOT NULL,
                score FLOAT,
                FULLTEXT INDEX fulltext_name(name)
                )ENGINE=MyISAM;
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t3\G
*************************** 1. row ***************************
      Table: t3
Create Table: CREATE TABLE 't3' (
  'id' int(11) NOT NULL,
  'name' varchar(20) COLLATE utf8_bin NOT NULL,
  'score' float DEFAULT NULL,
  FULLTEXT KEY 'fulltext_name' ('name')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，name 字段上已经建立了一个名为 fulltext_name 的全文索引。需要注意的是，由于目前只有 MyISAM 存储引擎支持全文索引，InnoDB 存储引擎还不支持全文索引，因此，在建立全文索引时，一定要注意表存储引擎的类型，对于经常需要索引的字符串、文字数据等信息，可以考虑存储到 MyISAM 存储引擎的表中。

4、创建单列索引

【例 2-26】创建一个表名为 t4 的表，在表中的 name 字段上建立索引名为 single_name 的单列索引，SQL 语句如下：

```
CREATE TABLE t4(id INT NOT NULL,
                name VARCHAR(20) NOT NULL,
                score FLOAT,
                INDEX single_name(name(20))
                );
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t4\G
*************************** 1. row ***************************
      Table: t4
Create Table: CREATE TABLE 't4' (
  'id' int(11) NOT NULL,
```

```
  'name' varchar(20) COLLATE utf8_bin NOT NULL,
  'score' float DEFAULT NULL,
  KEY 'single_name' ('name')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，name 字段上已经建立了一个名称为 single_name 的单列索引，并且索引的长度为 20。

5、创建多列索引

【例 2-27】创建一个表名为 t5 的表，在表中的 id 和 name 字段上建立索引名为 multi 的全文索引， SQL 语句如下：

```
CREATE TABLE t5(id INT NOT NULL,
                name VARCHAR(20) NOT NULL,
                score FLOAT,
                INDEX multi(id,name(20))
                );
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t5\G
*************************** 1. row ***************************
       Table: t5
Create Table: CREATE TABLE 't5' (
  'id' int(11) NOT NULL,
  'name' varchar(20) COLLATE utf8_bin NOT NULL,
  'score' float DEFAULT NULL,
  KEY 'multi' ('id','name')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，id 和 name 字段上已经建立了一个名为 multi 的多列索引。需要注意的是，在多列索引中，只有查询条件中使用了这些字段中的第一个字段时，多列索引才会被使用。为了验证这个说法是否正确，将 id 字段作为查询条件，通过 EXPLAIN 语句查看索引的使用情况，SQL 执行结果如下所示：

```
mysql> EXPLAIN SELECT * FROM t5 WHERE id=1 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t5
         type: ref
possible_keys: multi
          key: multi
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.09 sec)
```

从上述执行结果可以看出，possible_keys 和 key 的值都为 multi，说明 multi 索引已经存在，并且已经开始被使用了。但是，如果只使用 name 字段作为查询条件，SQL 执行结果如下所示：

```
mysql> EXPLAIN SELECT * FROM t5 WHERE name='Mike' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t5
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
        Extra: Using where
1 row in set (0.01 sec)
```

从上述执行结果可以看出，possible_keys 和 key 的值都为 NULL，说明 multi 索引还没有被使用。

6、创建空间索引

【例 2-28】创建一个表名为 t6 的表，在空间类型为 GEOMETRY 的字段上创建空间索引，SQL 语句如下：

```
CREATE TABLE t6(id INT,
                space GEOMETRY NOT NULL,
                SPATIAL INDEX sp(space)
                )ENGINE=MyISAM;
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t6\G
*************************** 1. row ***************************
       Table: t6
Create Table: CREATE TABLE 't6' (
  'id' int(11) DEFAULT NULL,
  'space' geometry NOT NULL,
  SPATIAL KEY 'sp' ('space')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，t6 表中的 space 字段上已经建立了一个名称为 sp 的空间索引。需要注意的是，创建空间索引时，所在字段的值不能为空值，并且表的存储引擎为 MyISAM。

二、使用 CREATE INDEX 语句在已经存在的表上创建索引

若想在一个已经存在的表上创建索引，可以使用 CREATE INDEX 语句，CREATE INDEX 语句创建索引的具体语法格式如下所示：

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX 索引名
 ON 表名 (字段名 [(长度)] [ASC|DESC]);
```

在上述语法格式中，UNIQUE、FULLTEXT 和 SPATIAL 都是可选参数，分别用于表示唯一性索引、全文索引和空间索引；INDEX 用于指明字段为索引。

为了便于大家学习如何使用 CREATE INDEX 语句在已经存在的表上创建索引，接下来，创建一个 book 表，该表中没有建立任何索引，创建 book 表的 SQL 语句如下所示：

```
CREATE TABLE book (
                    bookid  INT NOT NULL,
                    bookname VARCHAR(255) NOT NULL,
                    authors VARCHAR(255) NOT NULL,
                    info VARCHAR(255) NULL,
                    comment VARCHAR(255) NULL,
                    publicyear YEAR NOT NULL
                    );
```

创建好数据表 book 后，通过具体的案例为大家演示如何使用 CREAT INDEX 语句在已存在的数据表中创建索引，具体如下：

1、创建普通索引

【例 2-29】在 book 表中的 bookid 字段上建立一个名称为 index_id 的普通索引，SQL 语句如下所示：

```
CREATE INDEX index_id ON book(bookid);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  KEY 'index_id' ('bookid')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 bookid 字段上已经建立了一个名称为 index_id 的普通索引。

2、创建唯一性索引

【例 2-30】在 book 表中的 bookid 字段上建立一个名称为 uniqueidx 的唯一性索引，SQL 语句如下所示：

```
CREATE UNIQUE INDEX uniqueidx ON book(bookid);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
```

```
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  UNIQUE KEY 'uniqueidx' ('bookid'),
  KEY 'index_id' ('bookid')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 bookid 字段上已经建立了一个名称为 uniqueidx 的唯一性索引。

3、创建单列索引

【例 2-31】在 book 表中的 comment 字段上建立一个名称为 singleidx 的单列索引，SQL 语句如下所示：

```
CREATE INDEX singleidx ON book(comment);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  UNIQUE KEY 'uniqueidx' ('bookid'),
  KEY 'index_id' ('bookid'),
  KEY 'singleidx' ('comment')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 comment 字段上已经建立了一个名称为 singleidx 的单列索引。

4、创建多列索引

【例 2-32】在 book 表中的 authors 和 info 字段上建立一个名称为 mulitidx 的多列索引，SQL 语句如下所示：

```
CREATE INDEX mulitidx ON book(authors(20),info(20));
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  UNIQUE KEY 'uniqueidx' ('bookid'),
```

```
  KEY 'index_id' ('bookid'),
  KEY 'singleidx' ('comment'),
  KEY 'mulitidx' ('authors' (20), 'info' (20))
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 authors 和 info 字段上已经建立了一个名称为 mulitidx 的多列索引。

5、创建全文索引

【例 2-33】删除表 book，重新创建表 book，在表中的 info 字段上创建全文索引。

首先删除表 book，SQL 语句如下：

```
DROP TABLE book;
```

然后重新创建表 book，SQL 语句如下：

```
CREATE TABLE book (
                    bookid  INT NOT NULL,
                    bookname VARCHAR(255) NOT NULL,
                    authors VARCHAR(255) NOT NULL,
                    info VARCHAR(255) NULL,
                    comment VARCHAR(255) NULL,
                    publicyear YEAR NOT NULL
                  )ENGINE=MyISAM;
```

使用 CREATE INDEX 语句在 book 表的 info 字段上创建名称为 fulltextidx 的全文索引，SQL 语句如下：

```
CREATE FULLTEXT INDEX fulltextidx ON book(info);
```

为了验证全文索引 fulltextidx 是否创建成功，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  FULLTEXT KEY 'fulltextidx' ('info')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 info 字段上已经建立了一个名称为 fulltextidx 的全文索引。

6、创建空间索引

【例 2-34】创建表 t7，在表中的 g 字段上创建名称为 spatidx 的空间索引。

首先创建数据表 t7，SQL 语句如下：

```
CREATE TABLE t7(
                  g GEOMETRY NOT NULL
```

```
                )ENGINE=MyISAM;
```

使用 CREATE INDEX 语句在 t7 表的 g 字段上创建名称为 spatidx 的空间索引，SQL 语句如下：

```
CREATE SPATIAL INDEX spatidx ON t7(g);
```

为了验证空间索引 spatidx 是否创建成功，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  t7 \G
*************************** 1. row ***************************
       Table: t7
Create Table: CREATE TABLE 't7' (
  'g' geometry NOT NULL,
  SPATIAL KEY 'spatidx' ('g')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 g 字段上已经建立了一个名称为 spatidx 的空间索引。

三、使用 ALTER TABLE 语句在已经存在表上创建索引

在已经存在的表中创建索引，除了可以使用 CREATE INDEX 语句外，还可以使用 ALTER TABLE 语句。使用 ALTER TABLE 语句在已经存在表上创建索引的语法格式如下所示：

```
ALTER TABLE 表名 ADD [UNIQUE|FULLTEXT|SPATIAL]  INDEX
                    索引名 (字段名 [(长度)] [ASC|DESC])
```

在上述语法格式中，UNIQUE、FULLTEXT 和 SPATIAL 都是可选参数，分别用于表示唯一性索引、全文索引和空间索引，ADD 表示向表中添加字段。

接下来，同样以 book 表为例，对不同类型的索引进行详细讲解。为了使 book 表不包含任何索引，我们首先删除表 book，SQL 语句如下：

```
DROP TABLE book;
```

然后重新建立表 book，SQL 语句如下：

```
CREATE TABLE book (
                    bookid  INT NOT NULL,
                    bookname VARCHAR(255) NOT NULL,
                    authors VARCHAR(255) NOT NULL,
                    info VARCHAR(255) NULL,
                    comment VARCHAR(255) NULL,
                    publicyear YEAR NOT NULL
                    );
```

创建好数据表 book 后，就可以使用 ALTER TABLE 语句在已存在的数据表中创建索引了，具体如下：

1、创建普通索引

【例 2-35】在表中的 bookid 字段上创建名称为 index_id 的普通索引，SQL 语句如下：

```
ALTER TABLE book ADD INDEX index_id(bookid);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
```

```
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  KEY 'index_id' ('bookid')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 bookid 字段上已经建立了一个名称为 index_id 的普通索引。

2、创建唯一性索引

【例 2-36】在 book 表中的 bookid 字段上建立一个名称为 uniqueidx 的唯一性索引，SQL 语句如下：

```
ALTER TABLE book ADD UNIQUE uniqueidx(bookid);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  UNIQUE KEY 'uniqueidx' ('bookid'),
  KEY 'index_id' ('bookid')
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 bookid 字段上已经建立了一个名称为 uniqueidx 的唯一性索引。

3、创建单列索引

【例 2-37】在 book 表中的 comment 字段上建立一个名称为 singleidx 的单列索引，SQL 语句如下所示：

```
ALTER TABLE book ADD INDEX singleidx (comment(50));
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
```

```
  KEY 'index_id' ('bookid'),
  KEY 'singleidx' ('comment' (50))
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 comment 字段上已经建立了一个名称为 singleidx 的单列索引。

4、创建多列索引

【例 2-38】在 book 表中的 authors 和 info 字段上建立一个名称为 multidx 的多列索引，SQL 语句如下：

```
ALTER TABLE book ADD INDEX multidx(authors(20),info(50));
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  UNIQUE KEY 'uniqueidx' ('bookid'),
  KEY 'index_id' ('bookid'),
  KEY 'singleidx' ('comment' (50)),
  KEY 'multidx' ('authors' (20), 'info' (50))
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.02 sec)
```

从上述结果可以看出，book 表中的 authors 和 info 字段上已经建立了一个名称为 multidx 的多列索引。

5、创建全文索引

【例 2-39】删除表 book，重新创建表 book，在表中的 info 字段上创建全文索引。

首先删除表 book，SQL 语句如下：

```
DROP TABLE book;
```

然后重新创建表 book，SQL 语句如下：

```
CREATE TABLE book (
                    bookid  INT NOT NULL,
                    bookname VARCHAR(255) NOT NULL,
                    authors VARCHAR(255) NOT NULL,
                    info VARCHAR(255) NULL,
                    comment VARCHAR(255) NULL,
                    publicyear YEAR NOT NULL
                   )ENGINE=MyISAM;
```

使用 ALTER TABLE 语句在 book 表的 info 字段上创建名称为 fulltextidx 的全文索引，SQL 语句如下：

```
ALTER TABLE book ADD FULLTEXT INDEX fulltextidx(info);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  FULLTEXT KEY 'fulltextidx' ('info')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，book 表中的 info 字段上已经建立了一个名称为 fulltextidx 的全文索引。

6、创建空间索引

【例 2-40】创建表 t8，在表中的 space 字段上创建名称为 spatidx 的空间索引。

首先得创建数据表 t8，SQL 语句如下所示：

```
CREATE TABLE t8(
                space GEOMETRY NOT NULL
              )ENGINE=MyISAM;
```

使用 ALTER TABLE 语句在 book 表的 space 字段上创建名称为 spatidx 的空间索引，SQL 语句如下所示：

```
ALTER TABLE t8 ADD SPATIAL INDEX spatidx(space);
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql>  SHOW CREATE TABLE  t8 \G
*************************** 1. row ***************************
       Table: t8
Create Table: CREATE TABLE 't8' (
  'space' geometry NOT NULL,
  SPATIAL KEY 'spatidx' ('space')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，t8 表中的 space 字段上已经建立了一个名称为 spatidx 的空间索引。

2.6.3 删除索引

由于索引会占用一定的磁盘空间，因此，为了避免影响数据库性能，应该及时删除不再使用的索引。删除索引的方式有两种，具体如下：

1、使用 ALTER TABLE 删除索引

使用 ALTER TABLE 删除索引的基本语法格式如下所示：

```
ALTER TABLE 表名 DROP INDEX 字段名
```

【例 2-41】删除表 book 中名称为 fulltextidx 的全文索引。

在删除索引之前，首先通过 SHOW CREATE TABLE 语句查看 book 表，结果如下：

```
mysql> SHOW CREATE TABLE  book\G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL,
  FULLTEXT KEY 'fulltextidx' ('info')
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

从上述结果可以看出，表 book 中存在一个名称为 fulltextidx 的全文索引，要想删除该索引，可以使用以下 SQL 语句：

```
ALTER TABLE book DROP INDEX fulltextidx;
```

上述 SQL 语句执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE  book\G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE 'book' (
  'bookid' int(11) NOT NULL,
  'bookname' varchar(255) COLLATE utf8_bin NOT NULL,
  'authors' varchar(255) COLLATE utf8_bin NOT NULL,
  'info' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'comment' varchar(255) COLLATE utf8_bin DEFAULT NULL,
  'publicyear' year(4) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
1 row in set (0.00 sec)
```

由此可以看出，book 表中名称为 fulltextidx 的索引被成功删除了。

2、使用 DROP INDEX 删除索引

使用 DROP INDEX 删除索引的基本语法格式如下所示：

```
DROP INDEX 索引名 ON 表名;
```

【例 2-42】删除表 t8 中名称为 spatidx 的空间索引，SQL 语句如下：

```
DROP INDEX spatidx ON t8;
```

上述 SQL 代码执行后，使用 SHOW CREATE TABLE 语句查看表的结构，结果如下所示：

```
mysql> SHOW CREATE TABLE t8\G
*************************** 1. row ***************************
       Table: t8
Create Table: CREATE TABLE 't8' (
  'space' geometry NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin
```

```
1 row in set (0.00 sec)
```

由此可以看出，表 t8 中名称为 spatidx 的索引被成功删除了。

2.7 本章小结

本章主要讲解了数据库的基本操作、数据表的基本操作、数据类型、表的约束以及索引。其中，数据库和数据表的操作是本章的重要内容，需要大家通过实践练习加以透彻了解。表的约束和索引是本章难点，希望读者在使用时，可以结合表的实际情况去运用。