第 3 章

数据库的基本操作

数据库从字面上理解就是存储数据的仓库，就像超市里的商品仓库一样，不同类别的商品放在不同的库中，比如将办公用品存放到办公用品库。在 MySQL 服务器中的数据库可以有多个，分别存储不同的数据。要想将数据存储到数据库中，首先需要创建一个数据库。比如，把学生信息存放到学生信息数据库中，把员工信息存放到员工信息数据库中。本章主要讲解创建、查看、修改和删除数据库等基本操作。

3.1 创建数据库

启动 MySQL 服务后，使用 MySQL 自带的客户端软件输入正确的密码或者使用第三方数据库管理软件 Navicat 均可连接到数据库服务器（具体操作详见 2.4.3 节内容）。创建数据库是指在数据库系统中划分一块空间，用来存储相应的数据。这是进行表操作的基础，也是进行数据库管理的基础。本节主要使用两种方法：利用 SQL 语句创建数据库以及使用图形界面创建数据库。

3.1.1 创建数据库的基本语法

在 MySQL 数据库中存在系统数据库和自定义数据库，系统数据库是在安装 MySQL 后系统自带的数据库；自定义数据库是通过语法或图形操作界面工具由用户定义创建的数据库。在创建数据库之前，要先了解目前在 MySQL 数据库中已经存在哪些数据库。查看现有数据库的语句如下：

```
SHOW DATABASES;
```

使用上面的语句查看数据库的结果如图 3-1 所示。

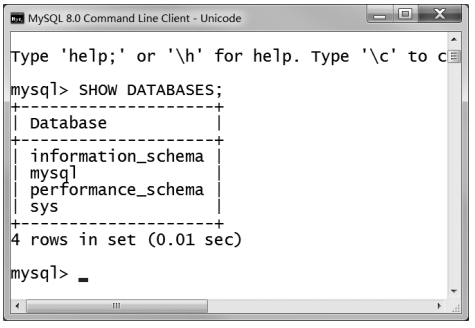

图 3-1 查看 MySQL 中的数据库

现在已经知道了如何查看已经存在的数据库，那么就可以创建新的用户自定义数据库了，创建数据库的语法如下所示：

```
CREATE DATABASE [IF NOT EXISTS]数据库名 CHARACTER SET 字符集 COLLATE collation_name;
```

其中：

(1) IF NOT EXISTS 表示当指定的数据库不存在时执行创建操作，否则忽略此操作。语法内使用“[]”括起来的选项表示可选参数。后续章节中出现的 SQL 语法结构中的“[]”都表示可选项。

(2) “数据库名”是唯一的，不能与其他数据库重名，否则将发生错误。数据库命名不要使用数字开头，可以是字母、数字、下画线(_)和“$”组成的任意字符串，不建议用汉字，并且尽量要有实际意义。名称最长可为 64 个字符，而别名最多可长达 256 个字符。不能使用 MySQL 关键字作为数据库名和表名。例如：学生管理系统的数据库可以直接使用英语命名为“student”，或者用汉语拼音的首字母命名为“xs”。在 MySQL 中不区分大小写，默认情况下，Windows 下的数据库名和表名的大小写是不敏感的，而在 Linux 下数据库名和表名的大小写是敏感的。为了便于数据库在平台间进行移植，建议读者采用小写来定义数据库名和表名。

(3) 设置数据库的字符集，其目的是为了避免数据库中存储的数据出现乱码的情况。如果在创建数据库时不指定字符集，那么就使用系统的字符集。系统默认的字符集是 Server Default。除了系统的默认字符集外，还可以选择 UTF-8、BIG5、DEC8、GB2312 和 GBK 等。如果要在数据库中存放中文，最好使用 UTF8。在创建数据库时，可以省略设置字符集的语句，这样就会采用数据库默认的字符集。

要查看 MySQL 数据库中支持的字符集，可以使用 SHOW CHARACTER SET 语句。查看当前数据库中所支持字符集的结果如图 3-2 所示。

```
mysql> SHOW CHARACTER SET;
+----------+---------------------------------+---------------------+--------+
| Charset  | Description                     | Default collation   | Maxlen |
+----------+---------------------------------+---------------------+--------+
| armscii8 | ARMSCII-8 Armenian              | armscii8_general_ci |      1 |
| ascii    | US ASCII                        | ascii_general_ci    |      1 |
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |      2 |
| binary   | Binary pseudo charset           | binary              |      1 |
| cp1250   | Windows Central European        | cp1250_general_ci   |      1 |
| cp1251   | Windows Cyrillic                | cp1251_general_ci   |      1 |
| cp1256   | Windows Arabic                  | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic                  | cp1257_general_ci   |      1 |
| cp850    | DOS West European               | cp850_general_ci    |      1 |
| cp852    | DOS Central European            | cp852_general_ci    |      1 |
| cp866    | DOS Russian                     | cp866_general_ci    |      1 |
| cp932    | SJIS for Windows Japanese       | cp932_japanese_ci   |      2 |
| dec8     | DEC West European               | dec8_swedish_ci     |      1 |
| eucjpms  | UJIS for Windows Japanese       | eucjpms_japanese_ci |      3 |
| euckr    | EUC-KR Korean                   | euckr_korean_ci     |      2 |
| gb18030  | China National Standard GB18030 | gb18030_chinese_ci  |      4 |
| gb2312   | GB2312 Simplified Chinese       | gb2312_chinese_ci   |      2 |
| gbk      | GBK Simplified Chinese          | gbk_chinese_ci      |      2 |
| geostd8  | GEOSTD8 Georgian                | geostd8_general_ci  |      1 |
| greek    | ISO 8859-7 Greek                | greek_general_ci    |      1 |
| hebrew   | ISO 8859-8 Hebrew               | hebrew_general_ci   |      1 |
| hp8      | HP West European                | hp8_english_ci      |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak      | keybcs2_general_ci  |      1 |
| koi8r    | KOI8-R Relcom Russian           | koi8r_general_ci    |      1 |
| koi8u    | KOI8-U Ukrainian                | koi8u_general_ci    |      1 |
| latin1   | cp1252 West European            | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European     | latin2_general_ci   |      1 |
| latin5   | ISO 8859-9 Turkish              | latin5_turkish_ci   |      1 |
| latin7   | ISO 8859-13 Baltic              | latin7_general_ci   |      1 |
| macce    | Mac Central European            | macce_general_ci    |      1 |
| macroman | Mac West European               | macroman_general_ci |      1 |
| sjis     | Shift-JIS Japanese              | sjis_japanese_ci    |      2 |
| swe7     | 7bit Swedish                    | swe7_swedish_ci     |      1 |
| tis620   | TIS620 Thai                     | tis620_thai_ci      |      1 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |      2 |
| ujis     | EUC-JP Japanese                 | ujis_japanese_ci    |      3 |
| utf16    | UTF-16 Unicode                  | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode                | utf16le_general_ci  |      4 |
| utf32    | UTF-32 Unicode                  | utf32_general_ci    |      4 |
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |      3 |
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_0900_ai_ci  |      4 |
+----------+---------------------------------+---------------------+--------+
41 rows in set (0.00 sec)
```

图 3-2　MySQL 中支持的字符集

3.1.2　使用 SQL 语句创建数据库

不管是在 MySQL 自带的客户端软件 MySQL 8.0 Command Line Client 中还是在 Navicat 软件中都可以输入 SQL 语句并执行，这里选择后者进行演示。

(1) 连接数据库。连接数据库成功之后，会在窗口左侧显示自己设置的“连接”图标，此处为“DBconn”，双击该图标后，会出现如图 3-3 所示的目录结构。

(2) 创建一个 SQL 语句执行窗口。如图 3-4 所示，在工具栏中单击“查询”工具，然后单击“新建查询”，即可创建一个 SQL 语句执行窗口。

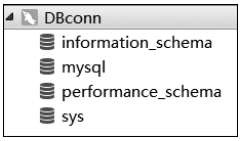

图 3-3　无数据库的目录结构图

图 3-4　创建 SQL 语句执行窗口

(3) 创建数据库。在刚刚创建的 SQL 语句执行窗口中输入创建数据库的 SQL 语句，其语法格式如下所示：

```
CREATE DATABASE db_name
```

其中，CREATE DATABASE 是创建数据库的固定格式，db_name 为要创建的数据库名称。接下来创建一个名为 db_test 的数据库，如例 3-1 所示。

在创建数据库时，如果不指定其使用的字符集或者字符集的校对规则，那么将根据 my.ini 文件中指定的 defaut-character-set 变量的值来设置其使用的字符集。从创建数据库的基本语法中可以看出，在创建数据库时，还可以指定数据库所使用的字符集。如果给“db_test”指定其字符集为 GBK，可以将命令改为：

```
CREATE DATABASE db_test CHARACTER SET=GBK;
```

【例 3-1】 使用 SQL 语句创建名为 db_test 的数据库。

```
CREATE DATABASE db_test;
```

执行结果如图 3-5 所示。

图 3-5 中已经标示出操作的具体步骤如下。

(1) 在窗口中写入创建数据库的 SQL 语句。

(2) 单击“运行”按钮，执行 SQL 语句。

(3) 如果下方的“信息”一栏中显示 OK，则表示创建数据库成功，否则创建失败。

(4) 右击 DBconn 图标后，选择“刷新”选项进行刷新操作。

(5) 刷新后会在左侧的目录结构中出现 db_test 数据库。在创建数据库后，MySQL 会在存储数据的 data 目录中创建一个与数据库同名的子目录(即 db_test)。因此，在 MySQL 中还可以通过在 data 下创建目录的方式完成数据库的创建。

注意：如果数据库已经存在，则不能创建成功，系统会提示“1007 - Can't create

图 3-5　使用 SQL 语句创建数据库

database 'db_test'; database exists"错误。在 MySQL 中,不允许同一系统中存在两个相同名称的数据库。为了避免类似的错误,可以将命令改写为:

```
CREATE DATABASE IF NOT EXISTS db_test;
```

还可以通过语法中给出的 CREATE SCHEMA 来创建数据库,两者的功能是一样的。比如,例 3-1 中使用 SQL 语句创建名为 db_test 的数据库的命令可以写为:

```
CREATES CHEMA db_test;
```

3.1.3　使用图形界面创建数据库

对于初学者而言,使用 SQL 语句创建数据库时,需要记住相应的 SQL 语句,与在图形界面上的操作相比比较困难,所以可以直接利用 Navicat 的图形界面创建数据库。

(1) 右击 DBconn,在弹出的列表中选择"新建数据库"选项,如图 3-6 所示。

图 3-6　选择"新建数据库"选项

(2) 在弹出的“新建数据库”窗口的“常规”栏中按照要求输入数据库名称、字符集和排序规则。其实只需要指定数据库名称即可,如果不指定字符集和排序规则,则表示选择默认配置,如图 3-7 所示。

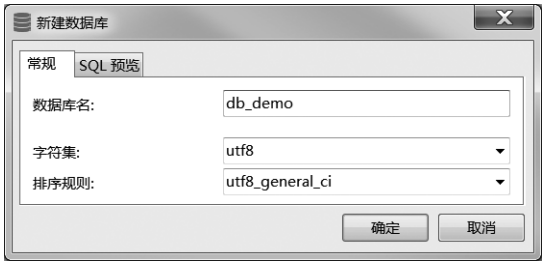

图 3-7　使用图形界面创建数据库

在“SQL 预览”栏中可以看到系统根据操作自动生成的 SQL 语句,如图 3-8 所示。

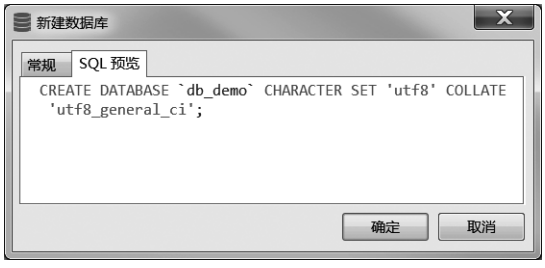

图 3-8　系统自动生成的 SQL 语句

(3) 单击“确定”按钮后,会在左侧的目录结构中看到名为 db_test 的数据库。

3.2　查看数据库

因为在创建数据库时,有可能出现“1007 - Can't create database 'db_test'; database exists”错误,所以希望在创建新的数据库之前先查看一下目前已有的数据库。这里同样介绍两种查看数据库的方法:使用 SQL 语句和图形界面。

3.2.1　使用 SQL 语句查看数据库

1. 查看所有的数据库

数据库创建完成后,若要查看该数据库的信息,或查看 MySQL 服务器当前都有哪些数据库,可以根据不同的需求选择以下的方式进行查看。在 SQL 语句执行窗口中输入如例 3-2 所示的 SQL 语句。

【例 3-2】 使用 SQL 语句查看所有数据库。

```
SHOW DATABASES;
```

执行结果如图 3-9 所示。

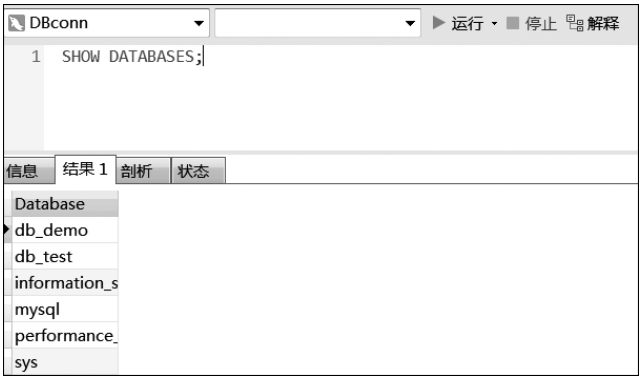

图 3-9　例 3-2 的执行结果

从图 3-9 中可以看到，除了在 3.1 节中创建的 db_test 和 db_demo 两个数据库外，还有 4 个 MySQL 数据库服务器自带的系统数据库。

(1) information_schema：提供了访问数据库元数据的方式(数据字典、所有表和库的结构信息)。其中保存着关于 MySQL 服务器所维护的所有其他数据库的信息，如数据库名、表名、列的数据类型和访问权限等。

(2) mysql：MySQL 的核心数据库，主要负责存储数据库的用户、权限设置、关键字以及 MySQL 自己需要使用的控制和管理信息等。

(3) performance_schema：主要用于收集数据库服务器性能参数，如提供进程等待的详细信息，包括锁、互斥变量和文件信息；保存历史的事件汇总信息，为提供 MySQL 服务器性能做出详细的判断；对于新增和删除监控事件点都非常容易，并可以改变 MySQL 服务器的监控周期等。

(4) sys：MySQL 5.7 新增的系统数据库，其在 MySQL 5.7 中是默认存在的，在 MySQL 5.6 及以上版本中可以手动导入。这个库通过视图的形式把 information_schema 和 performance_schema 结合起来，查询出令人更加容易理解的数据。库中所有的数据源都来自 performance_schema。目标是把 performance_schema 的复杂度降低，让 DBA 能更好地阅读这个库里的内容，以及更快地了解 DB 的运行情况。

对于初学者来说，建议不要随意删除和修改这些数据库，以避免造成服务器故障。

2. 查看指定的数据库

如果想要查看某个已经存在的数据库的创建信息，则需要使用的 SQL 语句的语法格式如下所示：

```
SHOW CREATE DATABASE db_name;
```

其中 SHOW CREATE DATABASE 为查看指定数据库的固定格式，db_name 为指定的数据库名称。接下来可以查看一下数据库 db_demo 的创建信息，如例 3-3 所示。

【例 3-3】 使用 SQL 语句查看指定的数据库。

```
SHOW CREATE DATABASE db_demo;
```

执行结果如图 3-10 所示。

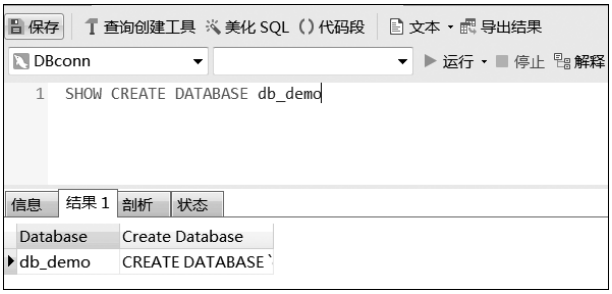

图 3-10　例 3-3 的执行结果

从图 3-10 中可以看到关于 db_demo 数据库的一些创建信息，如数据库名称，并且默认字符集为 UTF-8。其中“40100”表示版本号 4.1.00；“/ * 注释内容 * /”为 MySQL 支持的一种注释格式，并且 MySQL 对其进行了扩展，即当在注释中使用“!”加上版本号时，只要 MySQL 的当前版本等于或大于该版本号，则该注释中的 SQL 语句将被 MySQL 执行。但是这种方式只适用于 MySQL 数据库，不具有其他数据库的可移植性。

3.2.2　使用图形界面查看数据库

1. 查看所有的数据库

使用 Navicat 软件查看所有的数据库非常简单，所有的数据库都直接显示在左侧视图中，如图 3-11 所示(在查看之前可以先刷新列表)。

2. 查看指定的数据库

右击要查看的数据库 db_test，在弹出的列表中选择“编辑数据库”选项，之后会看到如图 3-12 所示的界面。

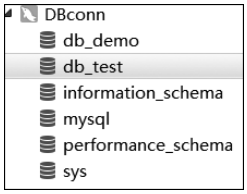

图 3-11　利用图形界面查看所有数据库

图 3-12　利用图形界面查看指定的数据库

从图 3-12 中可以清楚地看到关于数据库 db_test 的创建信息：数据库名称、字符集(utf8mb4)和排序规则(utf8_general_ci)。

注意：由于在创建数据库时并没有指定字符集和排序规则，所以 utf8mb4 和 utf8_general_ci 为 MySQL 默认的字符集和排序规则。

3.3 修改数据库

在 MySQL 数据库中，通过数据库修改语句只能对数据库使用的字符集进行修改，数据库中的这些特性存储在 db.opt 文件中。对于数据库使用的字符集，可以通过语句或图形界面来修改。

3.3.1 使用 SQL 语句修改数据库使用的字符集

修改数据库使用的字符集使用的是 ALTER 关键字，其语法格式如下所示：

```
ALTER DATABASE db_name CHARACTER SET new_charset;
```

其中 ALTER DATABASE 为修改数据库的固定语法格式，db_name 为要修改的数据库名称，CHARACTER SET 表示修改的是数据库的字符集，new_charset 为新的字符集名称。

假设要将 db_test 数据库的字符集由 UTF-8(默认字符集)修改为 GBK，具体操作如例 3-4 所示。

【例 3-4】 使用 SQL 语句修改所有数据库的字符集。

```
ALTER DATABASE db_test CHARACTER SET gbk;
```

执行结果如图 3-13 所示。

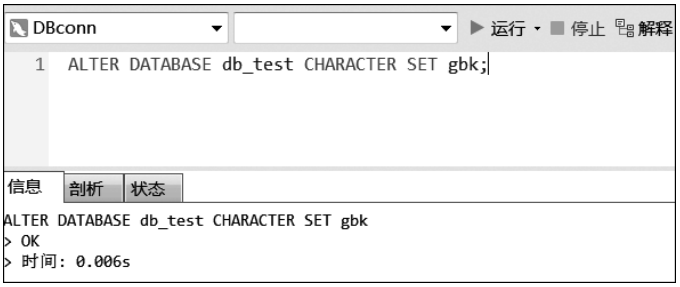

图 3-13 例 3-4 的执行结果

执行上述 SQL 语句后，显示 OK 则说明 SQL 语句执行成功。可以用 3.2.1 节中学到的知识查看 db_test 数据库是否修改成功，执行如下 SQL 语句：

```
SHOW CREATE DATABASE db_test;
```

执行结果如图 3-14 所示。

从图中可以清楚地看到，db_test 数据库的字符集已经被修改为 GBK。实际上，虽然只是将字符集修改为 GBK，但是其排序规则也会被自动修改为相应的 gbk_chinese_ci。

图 3-14　查看 db_test 字符集

3.3.2　使用图形界面修改数据库使用的字符集

使用 Navicat 软件来修改数据库的字符集是比较方便的，首先右击要修改的数据库，然后选择“编辑数据库”选项就可以进入“编辑数据库”窗口，在“字符集”下拉列表中选择合适的字符集，单击“确定”按钮即可。这里以 db_test 为例，将其字符集修改为 GBK，如图 3-15 所示。

图 3-15　利用图形界面修改数据库字符集

只需要选择要修改的字符集，排序规则无须做任何操作，因为系统会根据选择的字符集自动进行匹配。

3.4　选择数据库

由于 MySQL 服务器中的数据需要存储到数据表中，而数据表需要存储到对应的数据库中，并且 MySQL 服务器中又可以同时存在多个数据库，因此，在对数据和数据表进行操作前，首先需要选择数据库。基本语法格式如下：

```
USE 数据库名;
```

接下来选择数据库 db_test 进行操作，具体 SQL 语句与执行结果如下。

```
USE db_test;
Database changed
```

除了可以使用 USE 关键字选择数据库外，在用户登录 MySQL 服务器时也可以直接

选择要操作的数据库，基本语法格式如下：

```
mysql -h 主机 -u 用户名 -p 密码 数据库名;
```

上述语法中，在用户登录服务器的密码后添加要选择的数据库名称，按回车键后，MySQL 会在登录服务器后自动选择要操作的数据库。例如，密码为 123456 的 root 用户登录后要直接选择 db_test 数据库进行操作，具体 SQL 语句如下：

```
mysql -hlocalhost -u root -p 123456 db_test;
```

3.5 删除数据库

当数据库不再使用时应该将其删除，以确保数据库存储空间中存放的是有效的数据。删除数据库后就不能恢复该数据库了，因此不要轻易使用该操作。最好在删除数据库之前先将数据库进行备份，备份数据库的方法将在本书的后面章节中讲解。

但是，需要特别注意的是，数据库一旦删除，会将数据库中所有的表结构和数据一同删除，所以在做删除数据库的操作时需要非常慎重。

3.5.1 使用 SQL 语句删除数据库

在 MySQL 中删除数据库的语法是很简单的，只需要知道数据库的名称就可以将其删除掉，其语法格式如下所示：

```
DROP DATABASE db_name;
```

其中，DROP DATABASE 为删除数据库的固定语法格式，db_name 为要删除的数据库名称。以删除 db_test 数据库为例，具体操作如例 3-5 所示。

【例 3-5】 使用 SQL 语句删除数据库。

```
DROP DATABASE db_test;
```

执行结果如图 3-16 所示。

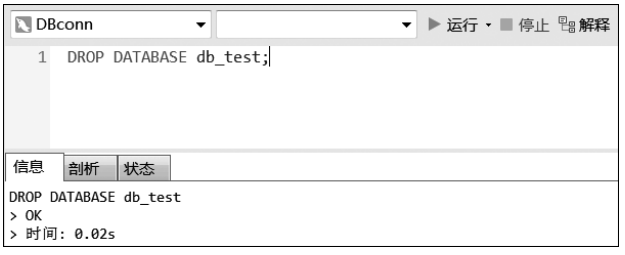

图 3-16 例 3-5 的执行结果

为了证实 db_test 数据库是否已经成功删除，可以执行查看所有数据库的 SQL 语句，如下所示：

```
SHOW DATABASES;
```

执行结果如图 3-17 所示。

图 3-17　查看 db_test 是否删除成功

图 3-17 中显示的数据库列表中并没有名为 db_test 的数据库，证明例 3-5 中的 SQL 语句确实执行成功。

3.5.2　使用图形界面删除数据库

使用 Navicat 软件删除数据库时，只需要右击要删除的数据库，然后在弹出的功能列表中选择“删除数据库”选项。下面以删除 db_demo 数据库为例，具体操作如图 3-18 所示。

单击“删除数据库”选项后，会出现如图 3-19 所示的“确认删除”对话框，如果确定要删除就单击“删除”按钮，否则单击“取消”按钮。

图 3-18　利用图形界面删除数据库

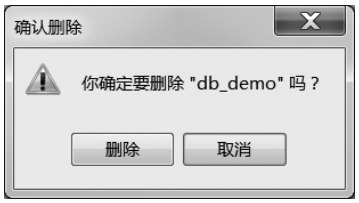

图 3-19　“确认删除”对话框

在执行删除数据库操作前，一定要备份需要保留的数据，确保数据的安全，以避免因误操作而造成严重的后果。

3.6 本章小结

本章主要介绍了创建、查看、修改、选择和删除数据库的知识。

3.7 思考与练习

1. 练习使用 SQL 语句创建数据库，数据库的名称为 School，字符集选择默认字符集 UTF-8。

2. 练习使用 SQL 语句查看 School 数据库是否创建成功，并查看其创建信息。

3. 练习使用 SQL 语句将 School 数据库的字符集修改为 GBK。

4. 练习使用 SQL 语句删除 School 数据库。

5. 查看数据库的语法格式是(　　)。

A. CREATE DATABASE 数据库名；　　B. SHOW DATABASES；

C. USE 数据库名；　　D. DROP DATABASE 数据库名；

6. SQL 代码“USE MyDB;”的功能是(　　)。

A. 修改数据库 MyDB　　B. 删除数据库 MyDB

C. 选择数据库 MyDB　　D. 创建数据库 MyDB

7. SQL 代码“DROP DATABASE MyDB001;”的功能是(　　)。

A. 修改数据库名为 MyDB001　　B. 删除数据库 MyDB001

C. 使用数据库 MyDB001　　D. 创建数据库 MyDB001

8. 下列描述错误的是(　　)。

A. 在 Windows 系统中，可以创建一个名称为 tb_bookInfo 的数据库和一个名称为 tb_bookinfo 的数据库

B. MySQL 数据库名可以由任意字母、阿拉伯数字、下画线(_)和“$”组成

C. MySQL 数据库名最长可为 64 个字符

D. 不能使用 MySQL 关键字作为数据库名和表名

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

A. CREATE DATABASE db_library；

B. SHOW db_library；

C. USE db_library；

D. SELECT db_library；

10. 下列关于修改数据库的描述中错误的是(　　)。

A. 使用 ALTER DATABASE 语句可以修改数据库名

B. 使用 ALTER DATABASE 的 CHARACTER SET 选项可以修改数据的字符集

C. 使用 ALTER DATABASE 的 COLLATE 选项可以指定字符集的校对规则

D. 使用 ALTER DATABASE 语句时可以不指定数据库名称

第 4 章

MySQL 支持的数据类型与运算符

数据类型作为数据的一种属性，能够表示数据所表达的信息以及存储类型。由于不同数据类型的存储方式不同，所以数据库中字段的数据类型对于数据库的优化非常重要。几乎所有的数据库（如 Oracle、MySQL 和 SQL Server 等）都定义了适用于自己的数据类型。但是，不同的数据库具有不同的特点，其定义的数据类型的种类和名称或多或少都会有所不同。

可以通过 MySQL 客户端命令行输入相关命令来查看 MySQL 数据库支持的所有数据类型。首先需要启动 MySQL 服务并登录数据库，然后输入 HELP DATA TYPES 命令，查看 MySQL 数据库支持的数据类型。

通过命令查看，可以看到 MySQL 支持多种数据类型，主要包括数值类型、日期和时间类型以及字符串类型三种。需要注意的是从 MySQL 5.7 开始，该数据库已经支持 JSON 数据的存储。本章将详细讲解各种数据类型。

通过本章的学习，读者可以了解算术运算符、比较运算符、逻辑运算符和位运算符等各种运算符的使用方法，还可以了解各种运算符的优先级别。在实际应用中经常需要使用运算符，学好本章可以让以后的操作更加简单。

4.1 数 值 类 型

MySQL 支持所有的 ANSI/ISO SQL 92 数字类型。数字分为整数和小数，其中整数用整数类型表示；小数用浮点数类型和定点数类型表示。例如可以将学生的年龄设置为整数类型，并将学生的成绩设置为浮点数类型等。不同的数据库对 SQL 标准做了不同的拓展。MySQL 除了支持 SQL 标准中的数值类型，如严格数值类型以及近似数值类型，还拓展了新的数值类型，如 BIT 等。

4.1.1 整数类型

顾名思义，整数类型是用来存储整数的。MySQL 支持的整数类型有 SQL 标准中的整数类型 INTEGER 和 SMALLINT，并在此基础上拓展了新的整数类型，如 TINYINT、MEDIUMINT 和 BIGINT。

由于不同的整数类型所占用的存储空间大小不同，所以表示的数据范围也不同。每种整数类型所占空间大小及表数范围如表 4-1 所示。

表 4-1 整数类型特性一览表

整数类型	大小	表数范围(有符号)	表数范围(无符号)	作 用
TINYINT	1 字节	−128～127	0～255	小整数值
SMALLINT	2 字节	−32768～32767	0～65535	大整数值
MEDIUMINT	3 字节	−8388608～8388607	0～16777215	大整数值
INT/INTEGER	4 字节	−2147483648～2147483647	0～4294967295	大整数值
BIGINT	8 字节	−9233372036854775808～9223372036854775807	0～18446744073709551615	极大整数值

从表 4-1 中可以看出，MySQL 主要支持 5 个整数类型，分别是 TINYINT、SMALLINT、MEDIUMINT、INT/INTEGER 和 BIGINT。这些类型除了存储空间大小和表数范围不同外，在很大程度上是相同的。

注意：

(1) INT 与 INTEGER 是同一种数据类型。

(2) 每种数据类型的表数范围可以根据所占字节数计算得出。

在选择数据类型时，要根据实际需求确定数值的范围，从而确定最合适的数据类型。如果选择不当，则可能会出现“Out of range”的错误提示。

下面使用命令 HELP INT 来查看一下 MySQL 中对于 INT 类型的描述，如例 4-1 所示。

【例 4-1】 查看 INT 数据类型。

```
mysql> HELP INT;
Name: 'INT'
Description:
INT[(M)][UNSIGNED][ZEROFILL]
```

在例 4-1 中，可以看到对于 INT 数据类型的描述中有三个可选属性。

(1) (M)：M 指定了 INT 型数据显示的宽度。MySQL 是以一个可选的显示宽度的形式对 SQL 标准进行扩展的。为了方便理解，这里举个例子，如果某个字段的数据类型为 INT(4)，则当存储的数据是 10 时(此处配合 ZEROFILL 使用)，则会在左边填补两个 0 凑足四位数；当存储的数据是 100 时，只需要在左边填补一个 0 即可；而当存储的数据是 100000 时，由于已经超过 4 位数，所以按照原样输出，即宽度会自动扩充。

(2) UNSIGNED：UNSIGNED(无符号)修饰符规定字段的值只能保存正数。由于不需要保存数字的正、负符号，所以在存储时可以节约一个“位”的空间，从而增大这个字段可以存储数值的范围。

(3) ZEROFILL：ZEROFILL(零填充)修饰符规定可以用 0(不是空格)来填补输出的值。使用这个修饰符可以阻止 MySQL 数据库存储负值。

注意：

- M 只是指定了预期的显示宽度，并不影响该字段所选取的数据类型的存储空间大

小以及表数范围。

- 如果要存储的数据长度不足显示宽度(M),则需要配合使用 ZEROFILL 修饰符才会填补 0。
- 如果某个字段使用了 ZEROFILL 修饰,则该字段会默认添加 UNSIGNED 修饰符。
- 在例 4-1 中,查看 MySQL 支持的数据类型中有一种称为 AUTO_INCREMENT 的类型。该类型可以看作是整数类型的一种属性,用于需要产生唯一标识符或者顺序值时。AUTO_INCREMENT 值一般从 1 开始,每行自动递增 1。

4.1.2　浮点数和定点数类型

如果想要在数据库中存储小数类型,则需要使用下面两种 MySQL 指出的数据类型:浮点数类型和定点数类型。浮点数类型在数据库中存放的是近似值,因此也称为近似值类型;定点数类型则在数据库中存放精确值。

浮点数类型包括 FLOAT(单精度)和 DOUBLE(双精度)两种,定点数类型只包括 DEC/DECIMAL/NUMERIC 一种(DEC/DECIMAL 与 NUMERIC 表示的是同一种数据类型,习惯上使用 DEC 或 DECIMAL)。

1. 浮点数类型

浮点数类型所占空间大小及表数范围如表 4-2 所示。

表 4-2　浮点数类型特性一览表

浮点数类型	大小	表数范围(有符号)	表数范围(无符号)	作　用
FLOAT	4 字节	(−3.402823466E+38, −1.175494351E−38)	0,(1.175494351E−38, 3.402823466E+38)	单精度浮点数值
DOUBLE	8 字节	(−1.7976931348623157E+308, −2.2250738585072014E−308)	0,(2.2250738585072014E−308, 1.7976931348623157E+308)	双精度浮点数值

从表中可以看出,DOUBLE 类型的精度要比 FLOAT 高。同样可以使用命令 HELP DOUBLE 查看 MySQL 中对于 DOUBLE 类型的描述,如例 4-2 所示。

【例 4-2】　查看 DOUBLE 数据类型。

```
mysql> HELP DOUBLE;
Name: 'DOUBLE'
Description:
DOUBLE[(M,D)][UNSIGNED][ZEROFILL]
```

从例 4-2 中可以看到,浮点数类型与整数类型类似,均有三个可选属性:(M, D)、UNSIGNED 和 ZEROFILL。其中 UNSIGNED 与 ZEROFILL 的含义与使用方法均已在 4.1.1 节中讲述过,所以在此重点讲解(M, D)。

(M, D)中的 M 表示浮点数据类型中数字的总个数,D 表示小数点后数字的个数。如果某字段定义为 DOUBLE(6, 3),而要存储的数据是 314.15926,则由于该数据小数

点后的位数超过 3，所以会在保存数据时四舍五入，从而使数据库中实际存放的是 314.15926 的近似值 314.159；如果要存放的数据是 3.1415926，则实际存放的是 3.142；但是当要存放的数据为 3141.5926 时，会提示“Out of range”错误。需要注意的是，与整数类型不一样的是，浮点数类型的宽度不会自动扩充。

FLOAT 和 DOUBLE 中的 M 和 D 如果没有指定值，则取值默认都为 0。在不超过数据类型的表数范围的情况下，并不会限制数字的总个数及小数点后数字的个数，即按照实际精度来显示。

如果要指定 M 和 D 值的话，也需要注意，M 和 D 的取值是有范围的。

(1) M 的取值范围为 0～255。但由于 FLOAT 只能保证 6 位有效数字的准确性，所以在 FLOAT(M,D)中，当 M＜＝6 时，数字通常是准确的；而 DOUBLE 只能保证 16 位有效数字的准确性，所以在 DOUBLE(M, D)中，当 M＜＝16 时，数字也通常是准确的。

(2) D 的取值范围为 0～30，同时必须满足 D＜＝M，否则会报错。

注意：浮点数类型(M, D)的用法为非标准用法，如果需要进行数据库迁移，则不要这么使用。

2. 定点数类型

定点数类型在数据库中是以字符串形式存储的，因此是精确值。定点数只有一种数据类型，即 DECIMAL，该数据类型用于精度要求非常高的计算中，如涉及金钱操作的领域。定点数所占内存大小及表数范围如表 4-3 所示。

表 4-3　定点数类型特性一览表

浮点数类型	大小	表数范围	作用
DECIMAL(M,D)	M＋2	最小和最大取值范围与 DOUBLE 相同； 指定 M 和 D 时，有效取值范围由 M 和 D 的大小决定	精度较高的小数值

【例 4-3】 查看 DECIMAL 数据类型。

```
mysql> HELP DECIMAL;
Name: 'DECIMAL'
Description:
DECIMAL[(M[,D])][UNSIGNED][ZEROFILL]
```

DECIMAL(M,D)的用法基本与浮点数类型(M,D)的用法相似，但是一些细节上仍有不同。

(1) DECIMAL 类型的 M 默认值为 10，D 默认值为 0。如果在创建表时，定义某字段为 DECIMAL 类型而没有带任何参数，则等同于 DECIMAL(10,0)，比如要存储的数据是 1.23，则保存到数据库中的实际是 1，而不是 1.23。如果只带一个参数，则该参数为 M 值，D 则取默认值 0。

(2) M 的取值范围为 1～65，取 0 时会被设为默认值 10，超出范围则会报错。

(3) D 的取值范围为 0～30，同时必须满足 D＜＝M，否则会报错。

4.1.3　BIT 类型

MySQL 5.0 以前，BIT 与 TINYINT 表示同一种数据类型。但是在 MySQL 5.0 以及之后的版本中，BIT 是一个完全不同的数据类型。可以使用 BIT 数据类型保存位字段值，即 BIT 可以方便地存储二进制数据。BIT 类型所占内存大小及表数范围如表 4-4 所示。

表 4-4　BIT 类型特性一览表

BIIT 类型	大小	表数范围	作用
BIT(M)	1～8 字节	BIT(1) ～BIT(64)	位字段值

从表 4-4 中可以看出，BIT 类型的表数范围与 M 有关，那 M 是什么呢？可以使用命令 HELP BIT 查看 MySQL 中对于 BIT 类型的描述，如例 4-4 所示。

【例 4-4】 查看 BIT 数据类型。

```
mysql> HELP BIT
Name: 'BIT'
Description:
BIT[(M)]
```

从例 4-4 中可以知道 M 指的是位数，取值范围为 1～64，如果没有指定 M 的值，则默认 M 为 1。如 BIT(1)的取值范围只有 0 和 1；BIT(4)的取值范围为 0～15；BIT(64)的取值范围为 $0 \sim 2^{64}-1$。

BIT 数据类型使用 b'value'的形式存储二进制数据，其中 value 指的是一个由 0 和 1 组成的二进制数据。如 b'111'和 b'10000000'分别表示十进制的 7 和 128。

如果 value 值的位数小于指定的 M，则会在 value 值的左侧补 0。如指定字段的数据类型为 BIT(6)，而存储的数据为 b'111'，则存入数据库中的数据实际为 b'000111'。

注意：在接下来创建表时，数字类型的选择应遵循如下原则。

(1) 选择最小的可用类型，如果该字段的值不会超过 127，则使用 TINYINT 比 INT 效果好。

(2) 对于完全都是数字的值，即无小数点时，可以选择整数类型，比如年龄。

(3) 浮点类型用于可能具有的小数部分的数，比如学生成绩。

(4) 在需要表示金额等货币类型时优先选择 DECIMAL 数据类型。

4.2　日期和时间类型

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 TIME、DATE、YEAR、DATETIME 和 TIMESTAMP。比如存放商场活动的持续时间和职员的出生日期等。从形式上来说，MySQL 日期类型的表示方法与字符串的表示方法相同(使用单引号括起来)。本质上，MySQL 日期类型的数据是一个数值类型，

可以参与简单的加、减运算。

五种日期与时间类型的取值范围及相应的“0”值如表 4-5 所示。

表 4-5 日期与时间类型特性一览表

类 型	格 式	取 值 范 围	0 值
TIME	'HH:MM:SS'	('-838:59:59', '838:59:59')	'00:00:00'
DATE	'YYYY-MM-DD'	('1000-01-01', '9999-12-31')	'0000-00-00'
YEAR	YYYY	(1901, 2155)	0000
DATETIME	'YYYY-MM-DD HH:MM:SS'	('1000-01-01 00:00:00', '9999-12-31 23:59:59')	'0000-00-00 00:00:00'
TIMESTAMP	'YYYY-MM-DD HH:MM:SS'	('1970-01-01 00:00:01' UTC, '2038-01-19 03:14:07' UTC)	'0000-00-00 00:00:00'

每种日期与时间类型都有一个取值范围和一个“0”值。在非严格模式下,当存储的数据格式不合法时,系统会给出警告,并将 0 值插入到数据库中。当插入的数据格式合法但是超出数据类型的范围时,该数据将被裁剪为范围最接近的端点(最大值或最小值)。但是在严格模式下,非法或合法但超出范围的数据是不允许存入数据库的,系统会提示错误。下面将在严格模式下讲解各种日期与时间类型。

注意:

(1) 严格模式:STRICT_TRANS_TABLES。该模式下如果插入的数据不合法或者超出范围均会提示错误,插入数据不会成功。

(2) 非严格模式:该模式下如果插入的数据不合法或者合法但超出范围,只会给出警告,但会插入成功,插入的数据为 0 值或者边界值。

(3) MySQL7 中默认为严格模式,如果想要修改,可以在 my.ini 配置文件中去掉 STRICT_TRANS_TABLES,并重新启动 MySQL 服务即可。

(4) 为了防止数据库迁移出现问题,建议使用严格模式。

4.2.1 TIME 类型

TIME 类型专门用来存储时间数据,如果不需要记录日期而只需要记录时间,则选择 TIME 类型是最合适的。

MySQL 中使用'HH:MM:SS'(如果所要表示的时间值较大,也可以使用'HHH:MM:SS')的形式来检索和显示 TIME 数据类型。其中 HH 表示小时,取值范围为 -838～838(因为 TIME 类型不仅可以表示一天中的某个时间,此时小时取值为 0～23;TIME 还可以表示两个事件的时间间隔,此时小时的取值可能会比 23 大,甚至是负数);MM 表示分,取值范围为 0～59;SS 表示秒,取值范围为 0～59。

可以使用下面这四种方式来指定 TIME 值。

(1) 'D HH:MM:SS[.fraction]'有分隔符格式的字符串。其中 D 表示天数,取值范围为 0～24;fraction 表示小数部分。比如指定数据类型为 TIME,要存储的值为'1 14:13:

12.8',则实际存储到数据库中的值为'38:13:13',这是因为在保存数据时,小时的值为(D * 24+HH),而 SS 后边的小数部分则会四舍五入(这是因为没有指定小数部分的位数,所以默认没有小数部分)。

但是,TIME 类型是支持存储小数部分的,这时需要指定数据类型为 TIME(fsp),其中 fsp(fractional seconds precision)为小数部分的位数,取值范围为 1～6。如指定数据类型为 TIME(3),要存储的数据为'14:13:12.8888',则存储到数据库中的数据为'14:13:12.889'。

其实没有必要严格按照上面的格式进行书写,还可以根据自己的需求任意省略其中的某些部分,像'D HH:MM:SS'、'D HH:MM'、'D HH'、'HH:MM:SS.fraction'、'HH:MM:SS'、'HH:MM'或'SS'这些非严格的语法也是正确的。

(2) 'HHMMSS[.fraction]': 无分隔符的字符串。如果是个有意义的时间值,如'101112',则会被解析为'10:11:12';但如果是个没有意义的时间值,如'109712'(非法时间值,其分钟部分的数值为 97,没有意义),则系统将提示"Incorrect time value"错误。

注意:

- HHMMSS[.fraction]格式中[]中的内容表示可选部分,以下类同。
- 在 MySQL 中,TIME 值'11:12'表示的是'11:12:00',而不是'00:11:12'。
- TIME 值'1112'表示的是'00:11:12'而不是'11:12:00';类似地,'12'表示的是'00:00:12'。
- 通过上述两个例子要注意区分有分隔符的'HH:MM:SS'TIME 格式和无分隔符的'HHMMSS'TIME 格式。
- 如果 TIME 值是'0'或者数字 0,则表示的是'00:00:00'。

(3) HHMMSS[.fraction]格式的数字。这种格式是以数字形式表示 TIME 数据的(注意,没有单引号)。如果该数字是个有意义的时间值,如 111213,则会被转换为标准时间格式的'11:12:13';如果该数字是个不合法的时间值,如 111267(该数据有不合法的秒数),则系统会提示"Incorrect time value"错误。如果直接输入数字 0,则会转换成 TIME 数据类型对应的 0 值,即'00:00:00'。

(4) 使用 CURRENT_TIME、NOW()或者 SYSDATE()三种方式获取系统当前时间。

4.2.2　DATE 类型

DATE 类型是专门用来存储日期数据的,如果只需要存储日期值而不需要时间部分,则应该选择 DATE 类型。

MySQL 中使用'YYYY-MM-DD'的形式来检索和显示 DATE 数据类型。其中 YYYY 表示年,取值范围为 1000～9999;MM 表示月,取值范围为 1～12;DD 表示日,取值范围为 1～31,还要根据实际的年份、月份确定具体的范围。如数据'2020-06-31'就是非法数据,因为 6 月没有 31 号。DATE 数据类型支持的范围是'1000-01-01'～'9999-12-31'。

可以使用下面这五种方式来指定 DATE 值。

(1) 'YYYY-MM-DD': 有分隔符格式的字符串。如果数据中月和日的值小于 10,则

不需要指定两位数，直接指定一位数即可。如 DATE 数据'2020-7-9'与'2020-07-09'表示的含义是相同的。

(2) 'YY-MM-DD'：有分隔符格式的字符串。其中 YY 的取值如果是 00～69，则年份自动转换为 2000～2069，如果是 70～99，则年份自动转换为 1970～1999。如果是'20-7-9'，会转换为'2020-07-09'；数据是'70-7-9'，则会转换为'1970-07-09'。

MySQL 中除了支持'YYYY-MM-DD'以及'YY-MM-DD'这种标准的分隔符格式外，还可以支持非标准的分隔符格式，即任何标点都可以用来作为间隔符，如'YYYY.MM.DD'、'YY.MM.DD'、'YYYY/MM/DD'、'YY/MM/DD'、'YYYY@MM@DD'和'YY@MM@DD'等表示的含义与'YYYY-MM-DD'或者'YY-MM-DD'相同。这是因为即使用户输入的 DATE 数据格式不严格，但是 MySQL 会将其转换成标准格式的数据然后保存到数据库中，如数据'2020@7@9'或者'20@7@9'均会被转换成'2020-07-09'。

(3) 'YYYYMMDD'或者'YYMMDD'无分隔符格式的字符串。如果 DATE 数据是个有意义的日期值，如'20200711'和'200711'均会被转换为'2020-07-11'；如果 DATE 数据是个不合法的日期值，如'201332'(其中的月和日部分无意义)，则系统会提示“Incorrect date value”错误。

(4) YYYYMMDD 或者 YYMMDD 格式的数字。这种格式是以数字形式表示 DATE 数据的(注意，没有单引号)。如果该数字是个有意义的日期值，如 20200711 和 200711 均会被转换为标准日期格式的'2020-07-11'；如果该数字是个不合法的日期值，如 201332，则系统会提示“Incorrect date value”错误。如果直接输入数字 0，则会转换成 DATE 数据类型对应的 0 值，即'0000-00-00'。用法与无分隔符格式的字符串表示形式基本一致。

(5) 使用 CURRENT_DATE、NOW()或 SYSDATE()三种方式获取系统当前日期。

4.2.3 YEAR 类型

YEAR 类型只是用来表示年份的数据类型，MySQL 中使用 YYYY 来检索和显示 YEAR 类型，其取值范围为 1901～2155 以及 0000。

YEAR 类型的使用较为简单，主要有以下四种方式。

(1) YYYY 或者'YYYY'格式的 4 位数字或字符串。使用该形式表示的年份范围为 1901～2155，具体写法如 2020、'2020'。但如果数据超出该范围，则会提示“Out of range”错误，如数据 2050 或'2050'。

(2) Y、YY 格式的 1～2 位数字。如果取值范围是 1～69，则将年份自动转换为 2001～2069；如果取值范围是 70～99，则将年份自动转换为 1970～1999；如果取值是 0，则年份会转换成 YEAR 类型对应的 0 值，即 0000。

(3) 'Y'、'YY'格式的 1～2 位字符串。如果取值范围是'0'～'69'，则将年份自动转换为 2000～2069；如果取值范围是'70'～'99'，则将年份自动转换为 1970～1999。

注意：数字表示的 0 与字符串表示的'0'或者'00'所表示的年份是不一样的：数字 0 表示的是 0000，而字符串'0'或者'00'表示的是 2000。

(4) 使用 NOW()或者 SYSDATE()两种方式获取系统当前年份。