第 3 章 表中数据的基本操作

本章学习目标

- 熟练掌握插入数据的方法；
- 熟练掌握修改数据的方法；
- 熟练掌握删除数据的方法。

通过第 2 章的学习，读者对数据库和数据表的基础操作有了一定了解，若是想操作数据库中的数据，还需进一步学习对数据的基本操作。本章将详细讲解对于数据库中的数据的基本操作，包括插入数据、更新数据和删除数据。

3.1 插 入 数 据

在实际应用中，用户可能会为表添加一条或多条数据，而插入数据的方式有多种。数据库既然是存储数据的地方，那么数据表中的数据是如何添加的呢？接下来将讲解向数据表中插入数据的方法，主要包括为所有字段插入数据、为指定列插入数据、批量插入数据等方式。

3.1.1 为所有字段插入数据

INSERT 语句有两种语法形式，一种是 INSERT…VALUES 语句，另一种是 INSERT…SET 语句。接下来讲解如何使用两种方式为所有字段插入数据。

1. 使用 INSERT…VALUES 语句

标准的 INSERT 语法要为每个插入值指定相应的字段。通过使用 INSERT 语句指定所有字段名可以向表中插入数据，标准的语法格式如下。

```
INSERT INTO 表名(字段名 1,字段名 2,……) VALUES(值 1,值 2,……);
```

其中，“字段名 1”和“字段名 2”是数据表中的字段名称，“值 1”和“值 2”是对应字段需要添加的数据，每个值的顺序、类型必须与字段名相对应。此处需要注意，除了数值和 NULL 值之外，字符、日期和时间数据类型的值必须使用单引号。

在讲解示例之前，首先在数据库 qf_test2 中创建一个员工表 emp，表结构如表 3.1 所示。

表 3.1 emp 表

字 段	数 据 类 型	说 明
id	INT	员工编号
name	VARCHAR(100)	员工姓名

续表

字　　段	数 据 类 型	说　　明
gender	VARCHAR(10)	员工性别
birthday	DATE	员工生日
salary	DECIMAL(10,2)	员工工资
entry_date	DATE	员工入职日期
resume_text	VARCHAR(200)	员工简介

首先,创建数据库 qf_test2,SQL 语句具体如下所示。

```
CREATE DATABASE qf_test2;
```

然后使用该数据库,具体如下所示。

```
mysql > USE qf_test2;
Database changed
```

接着创建数据表 emp,具体如下所示。

```
mysql > CREATE TABLE emp(
    -> id INT,
    -> name VARCHAR(100),
    -> gender VARCHAR(10),
    -> birthday DATE,
    -> salary DECIMAL(10,2),
    -> entry_date DATE,
    -> resume_text VARCHAR(200)
    -> );
Query OK, 0 rows affected (0.13 sec)
```

由上述结果可知,数据表创建完成。为了验证表 emp 是否被创建,使用 DESC 语句查看库中的 emp 表,具体如下所示。

```
mysql > DESC emp;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id          | int(11)      | YES  |     | NULL    |       |
| name        | varchar(100) | YES  |     | NULL    |       |
| gender      | varchar(10)  | YES  |     | NULL    |       |
| birthday    | date         | YES  |     | NULL    |       |
| salary      | decimal(10,2)| YES  |     | NULL    |       |
| entry_date  | date         | YES  |     | NULL    |       |
| resume_text | varchar(200) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

由上述结果可知,emp 表被成功创建。

下面通过具体示例演示使用 INSERT…VALUES 语句指定所有字段名并插入对应的值,如例 3-1 所示。

【例 3-1】 通过 INSERT…VALUES 语句插入数据。

```
mysql > INSERT INTO emp(
    -> id,name,gender,birthday,salary,entry_date,resume_text
    -> ) VALUES(
    -> 1,'lilei','male','1991 - 05 - 10',4000, '2013 - 06 - 10','none'
    -> );
Query OK, 1 row affected (0.07 sec)
```

由上述结果可知,插入数据完成。为了进一步验证,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday      | salary  | entry_date    | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
|  1 | lilei | male   | 1991 - 05 - 10 | 4000.00 | 2013 - 06 - 10 | none        |
+---+-----+------+-----------+-------+-----------+----------+
1 row in set (0.00 sec)
```

由上述结果可知,emp 表中的数据成功插入。因为表中只插入了一条记录,所以只查询到了一条结果。

emp 表中的数据具体如表 3.2 所示。

表 3.2 emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none

在插入数据时,INSERT 语句中的字段列表顺序并不一定要与表定义中的字段顺序相同,但 VALUES 中的值一定要和 INSERT 语句中的字段列表顺序对应。现在使用 INSERT…VALUES 语句不按表定义中的字段顺序插入数据,具体如下所示。

```
mysql > INSERT INTO emp(
    -> resume_text,entry_date,salary,birthday,gender,name,id
    -> ) VALUES(
    -> 'none','2014 - 10 - 20',6000,'1988 - 03 - 15','female','lucy',2
    -> );
Query OK, 1 row affected (0.03 sec)
```

由上述结果可知,插入数据完成。为了进一步验证,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday      | salary  | entry_date    | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
|  1 | lilei | male   | 1991 - 05 - 10 | 4000.00 | 2013 - 06 - 10 | none        |
|  2 | lucy  | female | 1988 - 03 - 15 | 6000.00 | 2014 - 10 - 20 | none        |
+---+-----+------+-----------+-------+-----------+----------+
2 rows in set (0.00 sec)
```

由上述结果可知,emp 表中的第二条数据成功插入。emp 表中的数据具体如表 3.3 所示。

表 3.3 emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none

由此可以看出,INSERT 语句中字段列表顺序可以与表定义的字段顺序不一致,但一般不建议这样做。

通常情况下,在 INSERT…VALUES 语句中为所有列插入数据时可以不指定字段名,其语法格式如下。

```
INSERT INTO 表名 VALUES (值 1,值 2,……);
```

其中,“值 1”和“值 2”表示每个字段需要添加的数据,每个值的顺序、类型必须与表定义的字段顺序、类型都一致。

下面通过具体示例演示在 INSERT…VALUES 语句中不指定字段名插入数据,如例 3-2 所示。

【例 3-2】 前面在数据库 qf_test2 中创建了员工表 emp,本例通过使用 INSERT…VALUES 语句不指定字段名的方式插入数据。

```
mysql> INSERT INTO emp VALUES(
    -> 3,'king','female','1993-06-15',7000,'2014-07-10','none'
    -> );
Query OK, 1 row affected (0.07 sec)
```

由上述结果可知,插入数据完成。为了验证数据是否被插入,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql> SELECT * FROM emp;
+----+-------+--------+------------+---------+------------+-------------+
| id | name  | gender | birthday   | salary  | entry_date | resume_text |
+----+-------+--------+------------+---------+------------+-------------+
|  1 | lilei | male   | 1991-05-10 | 4000.00 | 2013-06-10 | none        |
|  2 | lucy  | female | 1988-03-15 | 6000.00 | 2014-10-20 | none        |
|  3 | king  | female | 1993-06-15 | 7000.00 | 2014-07-10 | none        |
+----+-------+--------+------------+---------+------------+-------------+
3 rows in set (0.01 sec)
```

由上述结果可知,emp 表中的数据成功插入。emp 表中的数据具体如表 3.4 所示。

表 3.4 emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none

使用不指定字段名的方式来插入数据，VALUES 中值的顺序必须与表定义的字段顺序对应，否则会出现错误。接着进行错误演示，具体如下所示。

```
mysql > INSERT INTO emp VALUES(
    -> 'none','2013',5000,'1992 - 01 - 01','female','lilei',4
    -> );
Query OK, 1 row affected, 3 warnings (0.03 sec)
```

由上述结果可知，插入数据完成，与之前不同的是，在执行完成后有 3 个警告（3 warnings）。使用 SELECT 语句查看 emp 表中的数据，具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+-----------+
| id | name  | gender | birthday      | salary  | entry_date    | resume_text |
+---+-----+------+-----------+-------+-----------+-----------+
|  1 | lilei | male   | 1991 - 05 - 10 | 4000.00 | 2013 - 06 - 10 | none        |
|  2 | lucy  | female | 1988 - 03 - 15 | 6000.00 | 2014 - 10 - 20 | none        |
|  3 | king  | female | 1993 - 06 - 15 | 7000.00 | 2014 - 07 - 10 | none        |
|  0 | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4           |
+---+-----+------+-----------+-------+-----------+-----------+
4 rows in set (0.00 sec)
```

由上述结果可知，插入的第 4 条数据明显与字段不对应。emp 表中的数据具体如表 3.5 所示。

表 3.5　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4

2. 使用 INSERT…SET 语句

通过使用 INSERT…SET 语句指定所有字段名和对应的值可以向表中插入数据，其语法格式如下。

```
INSERT INTO 表名 SET 字段名 1 = 值 1,字段名 2 = 值 2,……;
```

其中，“字段名 1”和“字段名 2”是数据表中的字段名称，“值 1”和“值 2”是对应字段需要添加的数据，每个值的类型与字段名相对应。INSERT…SET 语句用于直接给表中的字段名指定对应的列值。实际上，在 SET 字句中指定要插入数据的字段名，等号后面为指定的数据，而对于未指定的字段名，列值为该字段的默认值。

下面通过具体示例演示在 INSERT…SET 语句中指定字段名插入数据，如例 3-3 所示。

【例 3-3】 前面在数据库 qf_test2 中创建了员工表 emp，本例通过使用 INSERT…SET 语句指定字段名的方式插入数据。

```
mysql > INSERT INTO emp SET
    -> id = 5,name = 'Mike',gender = 'male',birthday = '1996 - 09 - 18',salary = 6000,entry_date =
'2017 - 09 - 10',resume_text = 'none';
Query OK, 1 row affected (0.03 sec)
```

由上述结果可知,插入数据完成。为了验证数据是否被插入,使用 SELECT 语句查看 emp 表中的数据。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday    | salary  | entry_date   | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
|  1 | lilei | male   | 1991 - 05 - 10 | 4000.00 | 2013 - 06 - 10 | none        |
|  2 | lucy  | female | 1988 - 03 - 15 | 6000.00 | 2014 - 10 - 20 | none        |
|  3 | king  | female | 1993 - 06 - 15 | 7000.00 | 2014 - 07 - 10 | none        |
|  0 | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4           |
|  5 | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none        |
+---+-----+------+-----------+-------+-----------+----------+
3 rows in set (0.01 sec)
```

由上述结果可知,emp 表中的数据成功插入。emp 表中的数据具体如表 3.6 所示。

表 3.6　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4
5	Mike	male	1997-09-18	6000.00	2020-09-10	none

使用 INSERT…SET 语句可以指定所有字段插入数据,也可以指定部分字段插入数据,且方法相同,故在 3.1.2 节中不再讲解。需要注意的是,使用 INSERT…SET 语句的方式比较灵活。

3.1.2　为指定列插入数据

为指定列插入数据是指在一些情况下,只向部分字段插入数据,而其他字段的值为默认值即可。为指定列插入数据的语法格式如下。

```
INSERT INTO 表名(字段名 1,字段名 2,……) VALUES(值 1,值 2,……);
```

其中,“字段名 1”和“字段名 2”等表示数据表中的字段名称,“值 1”和“值 2”表示每个字段需要添加的数据,每个值的顺序、类型必须与字段名对应。

下面通过具体示例演示使用 INSERT 语句为指定列插入数据,具体如例 3-4 所示。

【例 3-4】 为 emp 表插入数据,且只插入前 4 个字段的数据。

```
mysql > INSERT INTO emp(
    -> id,name,gender,birthday
```

```
    -> ) VALUES(
    -> 6,'mary','female','1995-07-10'
    -> );
Query OK, 1 row affected (0.07 sec)
```

由上述结果可知，插入数据完成。为了进一步验证，使用 SELECT 语句查看 emp 表中的数据，具体如下所示。

```
mysql> SELECT * FROM emp;
+----+-------+--------+------------+--------+------------+-----------+
| id | name  | gender | birthday   | salary | entry_date | resume_text |
+----+-------+--------+------------+--------+------------+-----------+
|  1 | lilei | male   | 1991-05-10 | 4000.00 | 2013-06-10 | none      |
|  2 | lucy  | female | 1988-03-15 | 6000.00 | 2014-10-20 | none      |
|  3 | king  | female | 1993-06-15 | 7000.00 | 2014-07-10 | none      |
|  0 | 2013  | 5000   | 1992-01-01 | 0.00    | 0000-00-00 | 4         |
|  5 | Mike  | male   | 1997-09-18 | 6000.00 | 2020-09-10 | none      |
|  6 | mary  | female | 1995-07-10 | NULL    | NULL       | NULL      |
+----+-------+--------+------------+--------+------------+-----------+
5 rows in set (0.00 sec)
```

由上述结果可知，插入的第 6 条数据只有前 4 个字段有值，其他字段都是 NULL，即这些字段的默认值为 NULL。通过 SHOW CREATE TABLE 语句可以查看字段的默认值，具体如下所示。

```
mysql> SHOW CREATE TABLE emp\G;
*************************** 1. row ***************************
       Table: emp
Create Table: CREATE TABLE `emp` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(100) DEFAULT NULL,
  `gender` varchar(10) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `salary` decimal(10,2) DEFAULT NULL,
  `entry_date` date DEFAULT NULL,
  `resume_text` varchar(200) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
```

由上述结果可知，salary、entry_date 和 resume_text 字段的默认值都是 NULL。另外，为指定列添加数据时，指定字段无须与数据表中定义的顺序一致，只要和 VALUES 中的值顺序一致即可。emp 表中的数据具体如表 3.7 所示。

表 3.7　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4

续表

id	name	gender	birthday	salary	entry_date	resume_text
5	Mike	male	1997-09-18	6000.00	2020-09-10	none
6	mary	female	1995-07-10	NULL	NULL	NULL

接着演示指定的字段顺序与表定义的字段顺序不同时向 emp 插入数据,仍然只插入前 4 个字段的数据,具体如下所示。

```
mysql > INSERT INTO emp(
    -> birthday,gender,name,id
    -> ) VALUES(
    -> '1996 - 01 - 01','male','rin',7
    -> );
Query OK, 1 row affected (0.04 sec)
```

由上述结果可知,插入数据完成。为了进一步验证,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday    | salary | entry_date  | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
|  1 | lilei | male   | 1991 - 05 - 10 | 4000.00 | 2013 - 06 - 10 | none       |
|  2 | lucy  | female | 1988 - 03 - 15 | 6000.00 | 2014 - 10 - 20 | none       |
|  3 | king  | female | 1993 - 06 - 15 | 7000.00 | 2014 - 07 - 10 | none       |
|  0 | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4          |
|  5 | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none       |
|  6 | mary  | female | 1995 - 07 - 10 | NULL    | NULL         | NULL       |
|  7 | rin   | male   | 1996 - 01 - 01 | NULL    | NULL         | NULL       |
+---+-----+------+-----------+-------+-----------+----------+
6 rows in set (0.00 sec)
```

由上述结果可知,虽然指定字段的顺序和表定义的字段顺序不同,但指定字段的顺序和 VALUES 中的值对应,数据仍然插入成功。emp 表中的数据具体如表 3.8 所示。

表 3.8　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	4000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4
5	Mike	male	1997-09-18	6000.00	2020-09-10	none
6	mary	female	1995-07-10	NULL	NULL	NULL
7	rin	male	1996-01-01	NULL	NULL	NULL

3.1.3　批量插入数据

在实际开发中,用户可能会遇到向相同数据表中插入多条记录的情况,如果用 INSERT 语句一条一条地插入数据,显然是相当麻烦的。为了提高工作效率,用户可以选择批量插入

数据的方法。接下来讲解为所有列批量插入数据和为指定列批量插入数据。

1. 为所有列批量插入数据

事实上，使用一条 INSERT 语句就可以实现向数据库批量插入数据。与 3.1.2 节中插入一条数据类似，在批量插入时，语句中罗列多组 VALUES 对应的值即可，其语法格式如下。

```
INSERT INTO 表名[(字段名 1,字段名 2,……)]
VALUES(值 1,值 2,……),(值 1,值 2,……),……,(值 1,值 2,……);
```

其中，“字段名 1”和“字段名 2”表示数据表中的字段名称是可选的，“值 1”和“值 2”表示每个字段要添加的数据，每个值的顺序、类型必须与字段名对应。此处需要注意，每组数据要用括号括起来，每组括号用逗号间隔。

在讲解示例之前，首先在数据库 qf_test2 中创建一个教师表 teacher，如表 3.9 所示。

表 3.9　teacher 表

字　　段	数 据 类 型	说　　明
id	INT	教师编号
name	VARCHAR(50)	教师姓名
age	INT	教师年龄

首先使用数据库 qf_test2。

```
mysql > USE qf_test2;
Database changed
```

然后创建数据表 teacher，具体如下所示。

```
mysql > CREATE TABLE teacher(
    -> id INT,
    -> name VARCHAR(50),
    -> age INT
    ->);
Query OK, 0 rows affected (0.16 sec)
```

由上述结果可知，表被创建完成。为了验证表是否被创建，使用 DESC 语句查看库中的 teacher 表，具体如下所示。

```
mysql > DESC teacher;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| id    | int(11)   | YES  |     | NULL    |       |
| name  | varchar(50) | YES  |     | NULL    |       |
| age   | int(11)   | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
3 rows in set (0.06 sec)
```

由上述结果可知，teacher 表被成功创建。

下面通过具体示例演示如何为所有列批量插入数据，具体如例 3-5 所示。

【例 3-5】 通过 INSERT 语句为所有列批量插入数据。

```
mysql > INSERT INTO teacher(id,name,age)
    -> VALUES(1,'AA',20),(2,'BB',21);
Query OK, 2 rows affected (0.04 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

由上述结果可知,插入数据完成,通过一条 SQL 语句在表中添加了两条数据。

此处需要注意,通过 INSERT 语句同时插入多条记录时,MySQL 在返回结果中插入了一些之前返回结果中没有的信息,这些字符串的意思如下所示。

(1) Records:表明插入了几条记录。

(2) Duplicates:表明插入时被忽略的记录,可能是由于这些记录包含了重复的主键值。

(3) Warnings:表明有问题的主键值,如发生数据类型转换。

为了验证例 3-5 中的数据是否被添加,使用 SELECT 语句查看 teacher 表中的数据,具体如下所示。

```
mysql > SELECT * FROM teacher;
+---+----+-----+
| id | name | age  |
+---+----+-----+
|  1 | AA   |  20  |
|  2 | BB   |  21  |
+---+----+-----+
2 rows in set (0.00 sec)
```

由上述结果可知,teacher 表中的数据批量插入成功。

teacher 表中的数据具体如表 3.10 所示。

表 3.10 teacher 表中的数据

id	name	age
1	AA	20
2	BB	21

另外,SQL 语句中的字段名是可以省略的,下面演示省略字段名的情况,具体如下所示。

```
mysql > INSERT INTO teacher
    -> VALUES(3,'CC',22),(4,'DD',23);
Query OK, 2 rows affected (0.03 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

由上述结果可知,插入数据完成,在省略字段名的情况下,通过一条 SQL 语句插入了两条数据。为了验证数据是否被插入,使用 SELECT 语句查看 teacher 表中的数据,具体如下所示。

```
mysql > SELECT * FROM teacher;
+---+----+-----+
| id | name | age  |
+---+----+-----+
|  1 | AA   |  20  |
|  2 | BB   |  21  |
```

```
| 3 | CC   |  22  |
| 4 | DD   |  23  |
+---+------+------+
4 rows in set (0.00 sec)
```

由上述结果可知，teacher 表中的数据批量插入成功。teacher 表中的数据具体如表 3.11 所示。

表 3.11　teacher 表中的数据

id	name	age	id	name	age
1	AA	20	3	CC	22
2	BB	21	4	DD	23

2. 为指定列批量插入数据

在数据批量插入时，同样可以指定某几列，其他列自动使用默认值，这与 3.1.2 节中为指定列插入一条数据类似。

下面通过具体示例演示如何为指定列批量插入数据，具体如例 3-6 所示。

【例 3-6】 向 teacher 表批量插入数据，且只插入前两列数据。

```
mysql> INSERT INTO teacher(id,name)
    -> VALUES(5,'EE'),(6,'FF');
Query OK, 2 rows affected (0.13 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

由上述结果可知，插入数据完成，例 3-5 中只指定前两个字段，通过一条 SQL 语句插入了两条数据。为了验证数据是否被插入，使用 SELECT 语句查看 teacher 表中的数据，具体如下所示。

```
mysql> SELECT * FROM teacher;
+---+------+------+
| id | name | age  |
+---+------+------+
| 1 | AA   |  20  |
| 2 | BB   |  21  |
| 3 | CC   |  22  |
| 4 | DD   |  23  |
| 5 | EE   | NULL |
| 6 | FF   | NULL |
+---+------+------+
6 rows in set (0.00 sec)
```

由上述结果可知，teacher 表中的数据批量插入成功，第 3 列使用的是默认值。teacher 表中的数据具体如表 3.12 所示。

表 3.12　teacher 表中的数据

id	name	age	id	name	age
1	AA	20	4	DD	23
2	BB	21	5	EE	NULL
3	CC	22	6	FF	NULL

3.2 更新数据

前面讲解了如何插入数据,在数据插入之后,如果想变更数据,则需要更新数据表中的数据。在 MySQL 中使用 UPDATE 语句可以更新表中的数据,其语法格式如下。

```
UPDATE 表名
SET 字段名 1 = 值 1 [,字段名 2 = 值 2,……]
[WHERE 条件表达式];
```

其中,“字段名”用于指定需要更新的字段名称,“值”用于表示字段更新的新数据。如果要更新多个字段的值,可以用逗号分隔多个字段和值。“WHERE 条件表达式”是可选的,用于指定更新数据需要满足的条件。

UPDATE 语句可以更新表中的部分数据或全部数据,下面对这两种情况详细讲解。

1. 更新全部数据

当 UPDATE 语句中不使用 WHERE 条件语句时,会将表中所有数据的指定字段全部更新。

下面通过具体示例演示如何使用 UPDATE 语句更新全部数据,具体如例 3-7 所示。

【例 3-7】 将 teacher 表中的所有表示年龄的 age 字段更新为 30。

```
mysql > UPDATE teacher SET age = 30;
Query OK, 6 rows affected (0.04 sec)
Rows matched: 6 Changed: 6 Warnings: 0
```

由上述结果可知,执行完成后提示了“Changed: 6”,说明成功更新了 6 条数据。为了验证表中的数据是否被更新,使用 SELECT 语句查看 teacher 表中的数据,具体如下所示。

```
mysql > SELECT * FROM teacher;
+---+----+-----+
| id | name | age  |
+---+----+-----+
|  1 | AA   |  30  |
|  2 | BB   |  30  |
|  3 | CC   |  30  |
|  4 | DD   |  30  |
|  5 | EE   |  30  |
|  6 | FF   |  30  |
+---+----+-----+
6 rows in set (0.00 sec)
```

由上述结果可知,teacher 表中所有的 age 字段都更新为了 30,说明更新成功。teacher 表中的数据具体如表 3.13 所示。

表 3.13 teacher 表中的数据

id	name	age	id	name	age
1	AA	30	4	DD	30
2	BB	30	5	EE	30
3	CC	30	6	FF	30

2. 更新部分数据

在实际开发应用中，大多数需求是更新表中的部分数据，用户可以使用 UPDATE 语句修改基于 WHERE 子句的指定记录中的数据。

下面通过具体示例演示如何使用 UPDATE 语句更新部分数据，具体如例 3-8 所示。

【例 3-8】 将 emp 表中姓名为 lilei 的员工工资修改为 5000 元。

```
mysql > UPDATE emp SET salary = 5000 WHERE name = 'lilei';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

由上述结果可知，执行完成后提示了“Changed:1”，说明成功更新了一条数据。为了验证表中数据是否被更新，使用 SELECT 语句查看 emp 表中的数据，具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday      | salary  | entry_date    | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
| 1  | lilei | male   | 1991 - 05 - 10 | 5000.00 | 2013 - 06 - 10 | none        |
| 2  | lucy  | female | 1988 - 03 - 15 | 6000.00 | 2014 - 10 - 20 | none        |
| 3  | king  | female | 1993 - 06 - 15 | 7000.00 | 2014 - 07 - 10 | none        |
| 0  | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4           |
| 5  | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none        |
| 6  | mary  | female | 1995 - 07 - 10 | NULL    | NULL          | NULL        |
| 7  | rin   | male   | 1996 - 01 - 01 | NULL    | NULL          | NULL        |
+---+-----+------+-----------+-------+-----------+----------+
6 rows in set (0.00 sec)
```

由上述结果可知，emp 表中姓名为 lilei 的员工工资被成功修改为 5000 元。emp 表中的数据具体如表 3.14 所示。

表 3.14 emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	5000.00	2013-06-10	none
2	lucy	female	1988-03-15	6000.00	2014-10-20	none
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4
5	Mike	male	1997-09-18	6000.00	2020-09-10	none
6	mary	female	1995-07-10	NULL	NULL	NULL
7	rin	male	1996-01-01	NULL	NULL	NULL

接着将 emp 表中 id 为 2 的员工工资修改为 8000 元，将 resume_text 修改为 excellent。

```
mysql > UPDATE emp
    - > SET salary = 8000,resume_text = 'excellent'
    - > WHERE id = 2;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

上述结果提示“Changed: 1”，说明成功更新了一条数据。为了验证表中数据是否被更

新,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday    | salary  | entry_date   | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
| 1  | lilei | male   | 1991 - 05 - 10 | 5000.00 | 2013 - 06 - 10 | none       |
| 2  | lucy  | female | 1988 - 03 - 15 | 8000.00 | 2014 - 10 - 20 | excellent  |
| 3  | king  | female | 1993 - 06 - 15 | 7000.00 | 2014 - 07 - 10 | none       |
| 0  | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4          |
| 5  | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none       |
| 5  | mary  | female | 1995 - 07 - 10 | NULL    | NULL         | NULL       |
| 7  | rin   | male   | 1996 - 01 - 01 | NULL    | NULL         | NULL       |
+---+-----+------+-----------+-------+-----------+----------+
6 rows in set (0.00 sec)
```

由上述结果可知,emp 表中 id 为 2 的员工工资被成功修改为 8000,resume_text 被成功修改为 excellent。emp 表中的数据具体如表 3.15 所示。

表 3.15　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	5000.00	2013-06-10	none
2	lucy	female	1988-03-15	8000.00	2014-10-20	excellent
3	king	female	1993-06-15	7000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4
5	Mike	male	1997-09-18	6000.00	2020-09-10	none
6	mary	female	1995-07-10	NULL	NULL	NULL
7	rin	male	1996-01-01	NULL	NULL	NULL

接着将 emp 表中所有女性的工资在原有基础上增加 1000 元,具体如下所示。

```
mysql > UPDATE emp
    - > SET salary = salary + 1000
    - > WHERE gender = 'female';
Query OK, 2 rows affected (0.07 sec)
Rows matched: 2 Changed: 2 Warnings: 0
```

上述结果提示“Changed: 2”,说明成功更新了两条数据。为了验证表中数据是否被更新,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+-------+-----------+----------+
| id | name  | gender | birthday    | salary  | entry_date   | resume_text |
+---+-----+------+-----------+-------+-----------+----------+
| 1  | lilei | male   | 1991 - 05 - 10 | 5000.00 | 2013 - 06 - 10 | none       |
| 2  | lucy  | female | 1988 - 03 - 15 | 9000.00 | 2014 - 10 - 20 | excellent  |
| 3  | king  | female | 1993 - 06 - 15 | 8000.00 | 2014 - 07 - 10 | none       |
| 0  | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4          |
| 5  | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none       |
| 5  | mary  | male   | 1995 - 07 - 10 | NULL    | NULL         | NULL       |
| 7  | rin   | male   | 1996 - 01 - 01 | NULL    | NULL         | NULL       |
+---+-----+------+-----------+-------+-----------+----------+
6 rows in set (0.00 sec)
```

由上述结果可知，emp 表中所有 gender 字段值为 female 的员工的工资增加了 1000 元。emp 表中的数据具体如表 3.16 所示。

表 3.16　emp 表中的数据

id	name	gender	birthday	salary	entry_date	resume_text
1	lilei	male	1991-05-10	5000.00	2013-06-10	none
2	lucy	female	1988-03-15	9000.00	2014-10-20	excellent
3	king	female	1993-06-15	8000.00	2014-07-10	none
0	2013	5000	1992-01-01	0.00	0000-00-00	4
5	Mike	male	1997-09-18	6000.00	2020-09-10	none
6	mary	female	1995-07-10	NULL	NULL	NULL
7	rin	male	1996-01-01	NULL	NULL	NULL

3.3　删除数据

删除数据也是数据库的常见操作，如对于前面的员工表 emp，如果员工离职，那么需要从 emp 表中将离职的员工信息删除。本节将详细讲解如何在数据库中删除数据。

3.3.1　使用 DELETE 删除数据

DELETE 语句可能是 SQL 支持的所有数据修改语句中最简单的语句，使用 DELETE 语句删除表中的数据的语法格式如下。

```
DELETE FROM 表名 [WHERE 条件表达式];
```

其中，WHERE 条件语句是可选的，用于指定删除满足条件的数据。通过 DELETE 语句可以实现删除全部数据或删除部分数据，下面分别针对这两种情况进行详细地讲解。

1. 删除全部数据

当 DELETE 语句中不使用 WHERE 条件语句时，表中的所有数据将会被删除，具体如例 3-9 所示。

【例 3-9】 将 teacher 表中的所有数据都删除。

```
mysql > DELETE FROM teacher;
Query OK, 6 rows affected (0.03 sec)
```

由上述结果可知，数据删除完成。为了验证表中数据是否被删除，使用 SELECT 语句查看 teacher 表中的数据，具体如下所示。

```
mysql > SELECT * FROM teacher;
Empty set (0.00 sec)
```

由上述结果可知，teacher 表中数据为空，说明数据删除成功。

2. 删除部分数据

前面讲解了删除全部数据的方法，但在实际开发中的大多数需求是删除表中的部分数据，使用 WHERE 子句可以指定删除数据的条件，具体如例 3-10 所示。

【例 3-10】 将 emp 表中姓名为 rin 的员工记录删除。

```
mysql > DELETE FROM emp WHERE name = 'rin';
Query OK, 1 row affected (0.03 sec)
```

由上述结果可知,数据删除完成。为了验证数据是否被删除,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+--------+-----------+-----------+
| id | name  | gender | birthday      | salary   | entry_date    | resume_text |
+---+-----+------+-----------+--------+-----------+-----------+
| 1  | lilei | male   | 1991 - 05 - 10 | 5000.00 | 2013 - 06 - 10 | none        |
| 2  | lucy  | female | 1988 - 03 - 15 | 9000.00 | 2014 - 10 - 20 | excellent   |
| 3  | king  | female | 1993 - 06 - 15 | 8000.00 | 2014 - 07 - 10 | none        |
| 0  | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4           |
| 5  | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none        |
| 6  | mary  | male   | 1995 - 07 - 10 | NULL    | NULL          | NULL        |
+---+-----+------+-----------+--------+-----------+-----------+
5 rows in set (0.00 sec)
```

由上述结果可知,emp 表中姓名为 rin 的员工记录已经删除。

接着将 emp 表中工资小于 8500 的女员工删除,具体如下所示。

```
mysql > DELETE FROM emp
    - > WHERE gender = 'female' AND salary < 8500;
Query OK, 1 row affected (0.07 sec)
```

由上述结果可知,数据删除完成。为了验证数据是否被删除,使用 SELECT 语句查看 emp 表中的数据,具体如下所示。

```
mysql > SELECT * FROM emp;
+---+-----+------+-----------+--------+-----------+-----------+
| id | name  | gender | birthday      | salary   | entry_date    | resume_text |
+---+-----+------+-----------+--------+-----------+-----------+
| 1  | lilei | male   | 1991 - 05 - 10 | 5000.00 | 2013 - 06 - 10 | none        |
| 2  | lucy  | female | 1988 - 03 - 15 | 9000.00 | 2014 - 10 - 20 | excellent   |
| 0  | 2013  | 5000   | 1992 - 01 - 01 | 0.00    | 0000 - 00 - 00 | 4           |
| 5  | Mike  | male   | 1997 - 09 - 18 | 6000.00 | 2020 - 09 - 10 | none        |
| 6  | mary  | male   | 1995 - 07 - 10 | NULL    | NULL          | NULL        |
+---+-----+------+-----------+--------+-----------+-----------+
4 rows in set (0.00 sec)
```

由上述结果可知,emp 表中工资低于 8500 元的女员工记录已经被删除。

3.3.2 使用 TRUNCATE 删除数据

在 MySQL 中还可以用 TRUNCATE 语句删除表中的所有数据。TRUNCATE 是一个能够快速清空资料表内所有资料的 SQL 语法,能针对具有自动递增值的字段做计数重置归零并重新计算的作用。TRUNCATE 语句的语法格式如下。

```
TRUNCATE [TABLE] 表名;
```

如上所示的语法非常简单，TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同，二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。

接着通过具体示例演示如何使用 TRUNCATE 语句，具体如例 3-11 所示。

【例 3-11】 使用 TRUNCATE 语句将 emp 表中所有数据删除。

```
mysql > TRUNCATE TABLE emp;
Query OK, 0 rows affected (0.18 sec)
```

由上述结果可知，所有数据删除完成。为了验证数据是否被删除，使用 SELECT 语句查看 emp 表中的数据，具体如下所示。

```
mysql > SELECT * FROM emp;
Empty set (0.00 sec)
```

由上述结果可知，emp 表中全部数据被成功删除。

TRUNCATE 语句和 DELETE 语句都能实现删除表中的所有数据，但两者有一定的区别，它们的区别如下。

(1) TRUNCATE 语句是 DDL 语句，而 DELETE 语句是 DML 语句。

(2) TRUNCATE 语句只能作用于表，DELETE、DROP 语句可作用于表、视图等。

(3) TRUNCATE 语句用于删除表中所有的数据，而 DELETE 语句后面一般跟 WHERE 子句指定条件，用于删除部分数据。

(4) 使用 TRUNCATE 语句删除表中的数据后，再次向表中添加记录时，自增的字段默认值重置为 1；而使用 DELETE 语句删除表中的数据后，再次向表中添加记录时，自增的字段值为删除时该字段的最大值加 1。

MYSQL 中 TRUNCATE 和 DELETE 的区别如表 3.17 所示。

表 3.17 TRUNCATE 和 DELETE 的区别

	TRUNCATE	DELETE
条件删除	不支持	支持
事务回滚	不支持	支持
清理速度	快	慢
高水位重置	是	否

此处需要注意，当用户不再需要某个数据表时，用 DROP 语句；当用户仍要保留某个数据表，但要删除所有记录时，用 TRUNCATE 语句；当用户要删除部分记录时，用 DELETE 语句。

3.4 本章小结

本章主要针对如何操作数据表中的数据进行讲解，讲解了插入数据、更新数据和删除数据。希望读者掌握 MySQL 数据表中数据的基本操作，不要死记硬背，而应该多练习，在实

践中快速掌握所学知识。

3.5 习　　题

1. 填空题

(1) 向数据表中插入数据有多种方式,包括为所有列插入数据、为指定列插入数据、_____等。

(2) 在通常情况下,向数据表中插入数据应包括表中_____。

(3) 在使用 INSERT 语句为所有列插入数据时,也可以不指定_____。

(4) 使用_____语句可以实现数据的批量插入。

(5) 在 MySQL 中可以使用_____语句删除表中数据。

2. 思考题

(1) 简述如何为所有列插入数据。

(2) 简述如何为指定列插入数据。

(3) 简述如何批量插入数据。

(4) 简述如何更新指定列的数据。

(5) 简述 DELETE 和 TRUNCATE 的区别。

3.6 实验:电影心愿表的操作

1. 实验目的及要求

掌握使用 SQL 语句插入数据、修改数据、删除数据的操作。

2. 实验要求

(1) 在 mytest 数据库中创建一张电影心愿表(MovieWishTable)。

(2) 表中包括 4 个字段,分别为电影编号(id)、电影名(movie)、观看状态(status,1 表示已观看,0 表示未观看)、评分(score,1～10 分,0 表示未观看),表结构如表 3.18 所示。

表 3.18　电影心愿表结构

字　　段	数 据 类 型	说　　明
id	INT	电影编号
movie	VARCHAR(100)	电影名
status	INT(1)	观看状态,1 表示已观看,0 表示未观看
score	FlOAT(2)	评分

(3) 为电影心愿表插入 6 条数据,如表 3.19 所示。

表 3.19　电影心愿表数据信息

电影编号(id)	电影名(movie)	观看状态(status)	评分(score)
1	《放牛班的春天》	1	8.5
2	《当幸福来敲门》	0	0
3	《肖申克的救赎》	0	0

续表

电影编号(id)	电影名(movie)	观看状态(status)	评分(score)
4	《我和我的祖国》	1	9.6
5	《建党伟业》	1	9.2
6	《战狼 2》	1	8.5

(4) 将编号等于 2 的观看状态更新为 1,评分为 8.0。

(5) 删除编号等于 6 的电影心愿表信息。(DELETE、TRUNCATE)

3. 实验步骤

(1) 选择数据库。

若 mytest 数据库不存在,则创建 mytest 数据库,具体如下所示。

```
mysql > create database mytest;
Query OK, 1 row affected
```

若 mytest 数据库存在,则选择 mytest 数据库,具体如下所示。

```
mysql > use mytest;
Database changed
```

(2) 创建电影心愿表。

创建电影心愿表 MovieWishTable,具体操作如下所示。

```
mysql > create table MovieWishTable(id int,movie varchar(50),status int(1),score float(2));
Query OK, 0 rows affected
```

查看该表的表结构,具体如下所示。

```
mysql > desc MovieWishTable;
+-------+-----------+----+----+--------+-----+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+----+----+--------+-----+
| id    | int(11)   | YES  |     | NULL    |       |
| movie | varchar(50) | YES  |     | NULL    |       |
| status | int(1)   | YES  |     | NULL    |       |
| score | float     | YES  |     | NULL    |       |
+-------+-----------+----+----+--------+-----+
4 rows in set
```

(3) 插入数据。

根据实验需求向 MovieWishTable 表中插入 6 条数据,具体如下所示。

```
mysql > insert into MovieWishTable values
    -> (1,'«放牛班的春天»',1,8.5),
    -> (2,'«当幸福来敲门»',0,0),
    -> (3,'«肖申克的救赎»',0,0),
    -> (4,'«我和我的祖国»',1,9.6),
    -> (5,'«建党伟业»',1,9.2),
    -> (6,'«战狼 2»',1,8.5);
Query OK, 6 rows affected
Records: 6 Duplicates: 0 Warnings: 0
```

由上述结果可知,数据插入完成。使用 SELECT 语句验证表中数据,具体如下所示。

```
mysql > select * from MovieWishTable;
+---+--------------+------+-----+
| id | movie          | status | score |
+---+--------------+------+-----+
| 1 | «放牛班的春天» |    1 |  8.5 |
| 2 | «当幸福来敲门» |    0 |    0 |
| 3 | «肖申克的救赎» |    0 |    0 |
| 4 | «我和我的祖国» |    1 |  9.6 |
| 5 | «建党伟业»     |    1 |  9.2 |
| 6 | «战狼 2»       |    1 |  8.5 |
+---+--------------+------+-----+
6 rows in set
```

(4) 修改数据。

将编号等于 2 的电影观看状态更新为 1,评分设置为 8.0,具体如下所示。

```
mysql > update MovieWishTable set status = 1,score = 8.0
 where id = 2;
Query OK, 1 row affected
Rows matched: 1 Changed: 1 Warnings: 0
```

由上述结果可知,数据修改完成。使用 SELECT 语句验证 id=2 的信息,具体如下所示。

```
mysql > select * from MovieWishTable where id = 2;
+---+--------------+------+-----+
| id | movie          | status | score |
+---+--------------+------+-----+
| 2 | «当幸福来敲门» |    1 |    8 |
+---+--------------+------+-----+
```

(5) 删除数据。

此处使用 DELETE 语句删除编号等于 6 的电影心愿单信息,具体如下所示。

```
mysql > delete from MovieWishTable where id = 6;
Query OK, 1 row affected
```

由上述结果可知,数据删除完成,读者也可使用 TRUNCATE 语句进行删除数据。使用 SELECT 语句查看表中所有数据,具体如下所示。

```
mysql > select * from MovieWishTable;
+---+--------------+------+-----+
| id | movie          | status | score |
+---+--------------+------+-----+
| 1 | «放牛班的春天» |    1 |  8.5 |
| 2 | «当幸福来敲门» |    1 |    8 |
| 3 | «肖申克的救赎» |    0 |    0 |
| 4 | «我和我的祖国» |    1 |  9.6 |
| 5 | «建党伟业»     |    1 |  9.2 |
+---+--------------+------+-----+
5 rows in set
```