第 8 章

◂ 插入、更新与删除数据 ▸

学习目标 Objective

存储在系统中的数据是数据库管理系统（DBMS）的核心，数据库被设计用来管理数据的存储、访问和维护数据的完整性。MySQL 中提供了功能丰富的数据库管理语句，包括有效地向数据库中插入数据的 INSERT 语句，更新数据的 UPDATE 语句以及当数据不再使用时删除数据的 DELETE 语句。本章将详细介绍在 MySQL 中如何使用这些语句操作数据。

内容导航 Navigation

- 掌握如何向表中插入数据
- 掌握更新数据的方法
- 熟悉如何删除数据
- 掌握综合案例对数据表基本操作的方法和技巧

8.1 插入数据

在使用数据库之前，数据库中必须要有数据，MySQL 中使用 INSERT 语句向数据库表中插入新的数据记录。可以插入的方式有：插入完整的记录、插入记录的一部分、插入多条记录、插入另一个查询的结果，下面将分别介绍这些内容。

8.1.1 为表的所有字段插入数据

使用基本的 INSERT 语句插入数据，要求指定表名称和插入到新记录中的值。基本语法格式为：

```
INSERT INTO table_name (column_list) VALUES (value_list);
```

table_name 指定要插入数据的表名，column_list 指定要插入数据的那些列，value_list 指定每个列应对应插入的数据。注意，使用该语句时字段列和数据值的数量必须相同。

本章将使用样例表 person，创建语句如下：

```
CREATE TABLE person
(
id     INT UNSIGNED NOT NULL AUTO_INCREMENT,
name   CHAR(40) NOT NULL DEFAULT '',
age    INT NOT NULL DEFAULT 0,
info   CHAR(50) NULL,
PRIMARY KEY (id)
);
```

向表中所有字段插入值的方法有两种：一种是指定所有字段名，另一种是完全不指定字段名。

【例 8.1】在 person 表中，插入一条新记录，id 值为 1，name 值为 Green，age 值为 21，info 值为 Lawyer。

执行插入操作之前，使用 SELECT 语句查看表中的数据：

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

结果显示当前表为空，没有数据，接下来执行插入操作：

```
mysql> INSERT INTO person (id ,name, age , info)
    -> VALUES (1,'Green', 21, 'Lawyer');
Query OK, 1 row affected (0.00 sec)
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person;
+----+--------+-----+------------+
| id | name   | age | info       |
+----+--------+-----+------------+
|  1 | Green  |  21 | Lawyer     |
+----+--------+-----+------------+
```

可以看到插入记录成功。在插入数据时，指定了 person 表的所有字段，因此将为每一个字段插入新的值。

INSERT 语句后面的列名称顺序可以不是 person 表定义时的顺序。即插入数据时，不需要按照表定义的顺序插入，只要保证值的顺序与列字段的顺序相同就可以，如【例 8.2】所示。

【例 8.2】在 person 表中，插入一条新记录，id 值为 2，name 值为 Suse，age 值为 22，info 值为 dancer，SQL 语句如下：

```
mysql> INSERT INTO person (age ,name, id , info)
    -> VALUES (22, 'Suse', 2, 'dancer');
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person;
```

```
| id | name   | age | info      |
+----+--------+-----+-----------+
|  1 | Green  |  21 | Lawyer    |
|  2 | Suse   |  22 | dancer    |
+----+--------+-----+-----------+
```

由结果可以看到，INSERT 语句成功插入了一条记录。

使用 INSERT 插入数据时，允许列名称列表 column_list 为空，此时，值列表中需要为表的每一个字段指定值，并且值的顺序必须和数据表中字段定义时的顺序相同，如【例 8.3】所示。

【例 8.3】在 person 表中，插入一条新记录，id 值为 3，name 值为 Mary，age 值为 24，info 值为 Musician，SQL 语句如下：

```
mysql> INSERT INTO person
    -> VALUES (3,'Mary', 24, 'Musician');
Query OK, 1 row affected (0.00 sec)
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person;
+----+--------+-----+-----------+
| id | name   | age | info      |
+----+--------+-----+-----------+
|  1 | Green  |  21 | Lawyer    |
|  2 | Suse   |  22 | dancer    |
|  3 | Mary   |  24 | Musician  |
+----+--------+-----+-----------+
```

可以看到插入记录成功。数据库中增加了一条 id 为 3 的记录，其他字段值为指定的插入值。本例的 INSERT 语句中没有指定插入列表，只有一个值列表。在这种情况下，值列表为每一个字段列指定插入值，并且这些值的顺序必须和 person 表中字段定义的顺序相同。

虽然使用 INSERT 插入数据时可以忽略插入数据的列名称，但是值如果不包含列名称，那 VALUES 关键字后面的值不仅要求完整而且顺序必须和表定义时列的顺序相同。如果表的结构被修改，对列进行增加、删除或者位置改变操作，这些操作将使得用这种方式插入数据时的顺序也同时改变。如果指定列名称，则不会受到表结构改变的影响。

8.1.2 为表的指定字段插入数据

为表的指定字段插入数据，就是在 INSERT 语句中只向部分字段中插入值，而其他字段的值为表定义时的默认值。

【例 8.4】在 person 表中，插入一条新记录，name 值为 Willam，age 值为 20，info 值为 sports

man，SQL 语句如下：

```
mysql> INSERT INTO person (name, age,info)
    -> VALUES('Willam', 20, 'sports man');
Query OK, 1 row affected (0.00 sec)
```

提示信息表示插入一条记录成功。使用 SELECT 查询表中的记录，查询结果如下：

```
mysql> SELECT * FROM person;
+----+--------+-----+------------+
| id | name   | age | info       |
+----+--------+-----+------------+
|  1 | Green  |  21 | Lawyer     |
|  2 | Suse   |  22 | dancer     |
|  3 | Mary   |  24 | Musician   |
|  4 | Willam |  20 | sports man |
+----+--------+-----+------------+
```

可以看到插入记录成功。如查询结果显示，该 id 字段自动添加了一个整数值 4。在这里 id 字段为表的主键，不能为空，系统会自动为该字段插入自增的序列值。在插入记录时，如果某些字段没有指定插入值，MySQL 将插入该字段定义时的默认值。下面例子说明在没有指定列字段时，插入默认值。

【例 8.5】在 person 表中，插入一条新记录，name 值为 laura，age 值为 25，SQL 语句如下：

```
mysql> INSERT INTO person (name, age ) VALUES ('Laura', 25);
Query OK, 1 row affected (0.00 sec)
```

语句执行完毕，查看执行结果，

```
mysql> SELECT * FROM person;
+----+--------+-----+------------+
| id | name   | age | info       |
+----+--------+-----+------------+
|  1 | Green  |  21 | Lawyer     |
|  2 | Suse   |  22 | dancer     |
|  3 | Mary   |  24 | Musician   |
|  4 | Willam |  20 | sports man |
|  5 | Laura  |  25 | NULL       |
+----+--------+-----+------------+
```

可以看到，在本例插入语句中，没有指定 info 字段值，查询结果显示，info 字段在定义时默认为 NULL，因此系统自动为该字段插入空值。

要保证每个插入值的类型和对应列的数据类型匹配，如果类型不同，将无法插入，并且 MySQL 会产生错误。

8.1.3 同时插入多条记录

INSERT 语句可以同时向数据表中插入多条记录，插入时指定多个值列表，每个值列表之间用逗号分隔开，基本语法格式如下：

```
INSERT INTO table_name (column_list)
VALUES (value_list1), (value_list2),...,(value_listn);
```

value_list1,value_list2,…,value_listn;表示第 1,2,…,n 个插入记录的字段的值列表。

【例 8.6】在 person 表中，在 name、age 和 info 字段指定插入值，同时插入 3 条新记录，SQL 语句如下：

```
INSERT INTO person(name, age, info)
VALUES ('Evans',27, 'secretary'),
('Dale',22, 'cook'),
('Edison',28, 'singer');
```

语句执行结果如下：

```
mysql> INSERT INTO person(name, age, info)
    -> VALUES ('Evans',27, 'secretary'),
    -> ('Dale',22, 'cook'),
    -> ('Edison',28, 'singer');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

语句执行完毕，查看执行结果：

```
mysql>  SELECT * FROM person;
+----+----------+-----+-----------+
| id | name     | age | info      |
+----+----------+-----+-----------+
|  1 | Green    |  21 | Lawyer    |
|  2 | Suse     |  22 | dancer    |
|  3 | Mary     |  24 | Musician  |
|  4 | Willam   |  20 | sports man|
|  5 | Laura    |  25 | NULL      |
|  6 | Evans    |  27 | secretary |
|  7 | Dale     |  22 | cook      |
|  8 | Edison   |  28 | singer    |
+----+----------+-----+-----------+
```

由结果可以看到，INSERT 语句执行后，person 表中添加了 3 条记录，其 name 和 age 字段分别为指定的值，id 字段为 MySQL 添加的默认的自增值。

使用 INSERT 同时插入多条记录时，MySQL 会返回一些在执行单行插入时没有的额外信息，这些包含数的字符串的意思分别是：

- Records: 表明插入的记录条数。
- Duplicates: 表明插入时被忽略的记录，原因可能是这些记录包含了重复的主键值。
- Warnings: 表明有问题的数据值，例如发生数据类型转换。

【例 8.7】在 person 表中，不指定插入列表，同时插入 2 条新记录，SQL 语句如下：

```
INSERT INTO person
VALUES (9,'Harry',21, 'magician'),
(NULL,'Harriet',19, 'pianist');
```

语句执行结果如下：

```
mysql> INSERT INTO person
    -> VALUES (9,'Harry',21, 'magician'),
    -> (NULL,'Harriet',19, 'pianist');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person;
+----+-------------+-----+------------+
| id | name        | age | info       |
+----+-------------+-----+------------+
|  1 | Green       |  21 | Lawyer     |
|  2 | Suse        |  22 | dancer     |
|  3 | Mary        |  24 | Musician   |
|  4 | Willam      |  20 | sports man |
|  5 | Laura       |  25 | NULL       |
|  6 | Evans       |  27 | secretary  |
|  7 | Dale        |  22 | cook       |
|  8 | Edison      |  28 | singer     |
|  9 | Harry       |  21 | magician   |
| 10 | Harriet     |  19 | pianist    |
+----+-------------+-----+------------+
```

由结果可以看到，INSERT 语句执行后，person 表中添加了 2 条记录，与前面介绍单个 INSERT 语法不同，person 表名后面没有指定插入字段列表，因此，VALUES 关键字后面的多个值列表，都要为每一条记录的每一个字段列指定插入值，并且这些值的顺序必须和 person 表中字段定义的顺序相同，带有 AUTO_INCREMENT 属性的 id 字段插入 NULL 值，系统会自动为该字段插入唯一的自增编号。

一个同时插入多行记录的 INSERT 语句等同于多个单行插入的 INSERT 语句，但是多行的 INSERT 语句在处理过程中，效率更高。因为 MySQL 执行单条 INSERT 语句插入多行数据，比使用多条 INSERT 语句快，所以在插入多条记录时，最好选择使用单条 INSERT 语句的方式插入。

8.1.4 将查询结果插入到表中

INSERT 语句用来给数据表插入记录时，指定插入记录的列值。INSERT 还可以将 SELECT 语句查询的结果插入到表中，如果想要从另外一个表中合并个人信息到 person 表，不需要把每一条记录的值一个一个输入，只需要使用一条 INSERT 语句和一条 SELECT 语句组成的组合语句，即可快速地从一个或多个表中向一个表中插入多个行。基本语法格式如下：

```
INSERT INTO  table_name1  (column_list1)
SELECT (column_list2) FROM table_name2 WHERE (condition)
```

table_name1 指定待插入数据的表；column_list1 指定待插入表中要插入数据的哪些列；table_name2 指定插入数据是从哪个表中查询出来的；column_list2 指定数据来源表的查询列，该列表必须和 column_list1 列表中的字段个数相同，数据类型相同；condition 指定 SELECT 语句的查询条件。

【例 8.8】从 person_old 表中查询所有的记录，并将其插入到 person 表中。

首先，创建一个名为 person_old 的数据表，其表结构与 person 结构相同，SQL 语句如下：

```
CREATE TABLE person_old
(
id     INT UNSIGNED NOT NULL AUTO_INCREMENT,
name   CHAR(40) NOT NULL DEFAULT '',
age    INT NOT NULL DEFAULT 0,
info   CHAR(50) NULL,
PRIMARY KEY (id)
);
```

向 person_old 表中添加两条记录：

```
mysql> INSERT INTO person_old
    -> VALUES (11,'Harry',20, 'student'), (12,'Beckham',31, 'police');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM person_old;
+----+-------------+-----+---------+
| id | name     | age | info     |
+----+---------+------+---------+
| 11 | Harry   |  20  | student |
| 12 | Beckham |  31  | police   |
+----+----------+------+---------+
2 rows in set (0.02 sec)
```

可以看到，插入记录成功，peson_old 表中现在有两条记录。接下来将 person_old 表中所有的记录插入 person 表中，SQL 语句如下：

```
INSERT INTO person(id, name, age, info)
SELECT id, name, age, info FROM person_old;
```

语句执行结果如下：

```
mysql> INSERT INTO person(id, name, age, info)
    -> SELECT id, name, age, info FROM person_old;
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person;
+----+--------+-----+----------+
| id | name   | age | info     |
+----+--------+-----+----------+
|  1 | Green  |  21 | Lawyer   |
|  2 | Suse   |  22 | dancer   |
|  3 | Mary   |  24 | Musici   |
|  4 | Willam |  20 | sports   |
|  5 | Laura  |  25 | NULL     |
|  6 | Evans  |  27 | secret   |
|  7 | Dale   |  22 | cook     |
|  8 | Edison |  28 | singer   |
|  9 | Harry  |  21 | magici   |
| 10 | Harriet|  19 | pianis   |
| 11 | Harry  |  20 | student  |
| 12 | Beckham|  31 | police   |
+----+--------+-----+----------+
```

由结果可以看到，INSERT语句执行后，person表中多了两条记录，这两条记录和person_old表中的记录完全相同，数据转移成功。这里的 id 字段为自增的主键，在插入的时候要保证该字段值的唯一性，如果不能确定，可以插入的时候忽略该字段，只插入其他字段的值。

这个例子中使用的 person_old 表和 person 表的定义相同，事实上，MySQL 不关心 SELECT 返回的列名，它根据列的位置进行插入，SELECT 的第 1 列对应待插入表的第 1 列，第 2 列对应待插入表的第 2 列，等等。即使不同结果的表之间也可以方便地转移数据。

8.2 更新数据

表中有数据之后，接下来可以对数据进行更新操作，MySQL 中使用 UPDATE 语句更新表中的记录，可以更新特定的行或者同时更新所有的行。基本语法结构如下：

```
UPDATE table_name
SET column_name1 = value1,column_name2=value2,……,column_namen=valuen
WHERE (condition);
```

column_name1,column_name2,……,column_namen 为指定更新的字段的名称；value1, value2,……valuen 为相对应的指定字段的更新值；condition 指定更新的记录需要满足的条件。更新多个列时，每个“列-值”对之间用逗号隔开，最后一列之后不需要逗号。

【例 8.9】在 person 表中，更新 id 值为 11 的记录，将 age 字段值改为 15，将 name 字段值改为 LiMing，SQL 语句如下：

```
UPDATE person SET age = 15, name='LiMing' WHERE id = 11;
```

更新操作执行前可以使用 SELECT 语句查看当前的数据：

```
mysql> SELECT * FROM person WHERE id=11;
+----+-------+-----+---------+
| id | name  | age | info    |
+----+-------+-----+---------+
| 11 | Harry |  20 | student |
+----+-------+-----+---------+
```

由结果可以看到更新之前，id 等于 11 的记录的 name 字段值为 harry，age 字段值为 20，下面使用 UPDATE 语句更新数据，语句执行结果如下：

```
mysql> UPDATE person SET age = 15, name='LiMing' WHERE id = 11;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person WHERE id=11;
+----+--------+-----+---------+
| id | name   | age | info    |
+----+--------+-----+---------+
| 11 | LiMing |  15 | student |
+----+--------+-----+---------+
```

由结果可以看到，id 等于 11 的记录中的 name 和 age 字段的值已经成功被修改为指定值。

保证 UPDATE 以 WHERE 子句结束，通过 WHERE 子句指定被更新的记录所需要满足的条件，如果忽略 WHERE 子句，MySQL 将更新表中所有的行。

【例 8.10】在 person 表中，更新 age 值为 19~22 的记录，将 info 字段值都改为 student，SQL 语句如下：

```
UPDATE person SET info='student'  WHERE age  BETWEEN 19 AND 22;
```

更新操作执行前可以使用 SELECT 语句查看当前的数据：

```
mysql> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
+----+---------+-----+------------+
| id | name    | age | info       |
+----+---------+-----+------------+
|  1 | Green   |  21 | Lawyer     |
|  2 | Suse    |  22 | dancer     |
|  4 | Willam  |  20 | sports man |
|  7 | Dale    |  22 | cook       |
|  9 | Harry   |  21 | magician   |
| 10 | Harriet |  19 | pianist    |
+----+---------+-----+------------+
```

可以看到，这些 age 字段值在 19~22 的记录的 info 字段值各不相同。下面使用 UPDATE 语句更新数据，语句执行结果如下：

```
mysql> UPDATE person SET info='student' WHERE age BETWEEN 19 AND 22;
Query OK, 6 rows affected (0.00 sec)
Rows matched: 6  Changed: 6  Warnings: 0
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
+----+---------+-----+---------+
| id | name    | age | info    |
+----+---------+-----+---------+
|  1 | Green   |  21 | student |
|  2 | Suse    |  22 | student |
|  4 | Willam  |  20 | student |
|  7 | Dale    |  22 | student |
|  9 | Harry   |  21 | student |
| 10 | Harriet |  19 | student |
+----+---------+-----+---------+
```

由结果可以看到，UPDATE 执行后，成功将表中符合条件的 6 条记录的 info 字段值都改为 student。

8.3 删除数据

从数据表中删除数据使用 DELETE 语句，DELETE 语句允许 WHERE 子句指定删除条件。DELETE 语句基本语法格式如下：

```
DELETE FROM table_name [WHERE <condition>];
```

table_name 指定要执行删除操作的表；“[WHERE <condition>]”为可选参数，指定删除条件，如果没有 WHERE 子句，DELETE 语句将删除表中的所有记录。

【例 8.11】在 person 表中，删除 id 等于 11 的记录。

执行删除操作前，使用 SELECT 语句查看当前 id=11 的记录：

```
mysql> SELECT * FROM person WHERE id=11;
+----+--------+-----+---------+
| id | name   | age | info    |
+----+--------+-----+---------+
| 11 | LiMing |  15 | student |
+----+--------+-----+---------+
```

可以看到，现在表中有 id=11 的记录，下面使用 DELETE 语句删除该记录，语句执行结果如下：

```
mysql> DELETE FROM person WHERE id = 11;
Query OK, 1 row affected (0.02 sec)
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person WHERE id=11;
Empty set (0.00 sec)
```

查询结果为空，说明删除操作成功。

【例 8.12】在 person 表中，使用 DELETE 语句同时删除多条记录，在前面 UPDATE 语句中将 age 字段值在 19~22 的记录的 info 字段值修改为 student，在这里删除这些记录，SQL 语句如下：

```
DELETE FROM person WHERE age BETWEEN 19 AND 22;
```

执行删除操作前，使用 SELECT 语句查看当前的数据：

```
mysql> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
+----+-----------+-----+---------+
| id | name      | age | info    |
+----+-----------+-----+---------+
|  1 | Green     |  20 | student |
|  2 | Suse      |  21 | student |
|  4 | Willam    |  22 | student |
|  7 | Dale      |  22 | student |
|  9 | Harry     |  21 | student |
| 10 | Harriet   |  19 | student |
+----+-----------+-----+---------+
```

可以看到，这些 age 字段值在 19~22 的记录存储在表中。下面使用 DELETE 删除这些记录：

```
mysql> DELETE FROM person WHERE age BETWEEN 19 AND 22;
Query OK, 6 rows affected (0.00 sec)
```

语句执行完毕，查看执行结果：

```
mysql> SELECT * FROM person WHERE age BETWEEN 19 AND 22;
Empty set (0.00 sec)
```

查询结果为空，删除多条记录成功。

【例 8.13】删除 person 表中所有记录，SQL 语句如下：

```
DELETE FROM person;
```

执行删除操作前，使用 SELECT 语句查看当前的数据：

```
mysql> SELECT * FROM person;
+----+---------+-----+-----------+
| id | name    | age | info      |
+----+---------+-----+-----------+
|  3 | Mary    |  24 | Musician  |
|  5 | Laura   |  25 | NULL      |
|  6 | Evans   |  27 | secretary |
| 12 | Beckham |  31 | police    |
+----+---------+-----+-----------+
```

结果显示 person 表中还有 4 条记录，执行 DELETE 语句删除这 4 条记录：

```
mysql> DELETE FROM person;
Query OK, 4 rows affected (0.00 sec)
```

语句执行完毕，查看执行结果：

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

查询结果为空，删除表中所有记录成功，现在 person 表中已经没有任何数据记录。

如果想删除表中的所有记录，还可以使用 TRUNCATE TABLE 语句，TRUNCATE 将直接删除原来的表，并重新创建一个表，其语法结构为 TRUNCATE TABLE table_name。TRUNCATE 直接删除表而不是删除记录，因此执行速度比 DELETE 快。

8.4 综合案例——记录的插入、更新和删除

本章重点介绍了数据表中数据的插入、更新和删除操作。MySQL 中对数据的操作没有任何提示，因此在更新和删除数据时，一要谨慎小心，查询条件一定要准确，避免造成数据的丢

失。本章的综合案例包含了对数据表中数据的基本操作，包括记录的插入、更新和删除。

1. 案例目的

掌握表数据基本操作。创建表 books，对数据表进行插入、更新和删除操作。books 表结构以及表中的记录，如表 8.1 和表 8.2 所示。

表 8.1 books 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
id	书编号	INT(11)	是	否	是	是	否
name	书名	VARCHAR(50)	否	否	是	否	否
authors	作者	VARCHAR(100)	否	否	是	否	否
price	价格	FLOAT	否	否	是	否	否
pubdate	出版日期	YEAR	否	否	是	否	否
note	说明	VARCHAR(255)	否	否	否	否	否
num	库存	INT(11)	否	否	是	否	否

表 8.2 books 表中的记录

id	name	authors	price	pubdate	note	num
1	Tale of AAA	Dickes	23	1995	novel	11
2	EmmaT	Jane lura	35	1993	joke	22
3	Story of Jane	Jane Tim	40	2001	novel	0
4	Lovey Day	George Byron	20	2005	novel	30
5	Old Land	Honore Blade	30	2010	law	0
6	The Battle	Upton Sara	33	1999	medicine	40
7	Rose Hood	Richard Haggard	28	2008	cartoon	28

2. 案例操作过程

步骤 01 创建数据表 books，并按表 8.1 结构定义各个字段。

```
CREATE TABLE books
(
id       INT(11) NOT NULL PRIMARY KEY,
name     VARCHAR(50) NOT NULL,
authors  VARCHAR(100) NOT NULL,
price        FLOAT NOT NULL,
pubdate  YEAR NOT NULL,
note     VARCHAR(255) NULL,
num      INT(11) NOT NULL DEFAULT 0
);
```

步骤 02 将表 8.2 中的记录插入 books 表中，分别使用不同的方法插入记录，执行过程如下。

表创建好之后，使用 SELECT 语句查看表中的数据，结果如下：

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

可以看到，当前表中为空，没有任何数据，下面向表中插入记录。

（1）指定所有字段名称插入记录，SQL 语句如下。

```
mysql> INSERT INTO books
    -> (id, name, authors, price, pubdate,note,num)
    -> VALUES(1, 'Tale of AAA', 'Dickes', 23, '1995', 'novel',11);
Query OK, 1 row affected (0.02 sec)
```

语句执行成功，插入了一条记录。

（2）不指定字段名称插入记录，SQL 语句如下。

```
mysql> INSERT INTO books
    -> VALUES (2,'EmmaT','Jane lura',35,'1993', 'joke',22);
Query OK, 1 row affected (0.01 sec)
```

语句执行成功，插入了一条记录。

使用 SELECT 语句查看当前表中的数据：

```
mysql> SELECT * FROM books;
+----+-----------------+-----------+-------+---------+---------+-----+
| id | name            | authors   | price | pubdate | note    | num |
+----+-----------------+-----------+-------+---------+---------+-----+
|  1 | Tale of AAA | Dickes    |    23 |  1995 | novel   |  11 |
|  2 | EmmaT       | Jane lura |    35 |  1993 | joke    |  22 |
+----+-----------------+-----------+-------+---------+---------+-----+
2 rows in set (0.00 sec)
```

可以看到，两条语句分别成功插入了两条记录。

（3）同时插入多条记录。

使用 INSERT 语句将剩下的多条记录插入表中，SQL 语句如下：

```
mysql> INSERT INTO books
    -> VALUES(3, 'Story of Jane', 'Jane Tim', 40, '2001', 'novel', 0),
    -> (4, 'Lovey Day', 'George Byron', 20, '2005', 'novel', 30),
    -> (5, 'Old Land', 'Honore Blade', 30, '2010', 'law',0),
    -> (6,'The Battle','Upton Sara',33,'1999', 'medicine',40),
    -> (7,'Rose Hood','Richard Kale',28,'2008', 'cartoon',28);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

由结果可以看到，语句执行成功，总共插入了 5 条记录，使用 SELECT 语句查看表中所有的记录：

```
mysql> SELECT * FROM books;
+----+---------------------+-------------+---------+-------------+------
--------+-----+
| id | name           | authors   | price  | pubdate   | note      | num |
+----+---------------------+-------------+---------+-------------+-----
----------+-----+
|  1 | Tale of AAA    | Dickes     |    23 |    1995 | novel     |  11 |
|  2 | EmmaT          | Jane lura    |    35 |    1993 | joke      |  22 |
|  3 | Story of Jane    | Jane Tim     |    40 |    2001 | novel     |   0 |
|  4 | Lovey Day      | George Byron |    20 |    2005 | novel     |  30 |
|  5 | Old Land       | Honore Blade |    30 |    2010 | law       |   0 |
|  6 | The Battle     | Upton Sara   |    33 |    1999 | medicine  |  40 |
|  7 | Rose Hood      | Richard Kale |    28 |    2008 | cartoon   |  28 |
+----+---------------------+-------------+---------+-------------+-
------------+-----+
7 rows in set (0.00 sec)
```

由结果可以看到，所有记录成功插入表中。

步骤03 将小说类型（novel）的书的价格都增加5。

执行该操作的SQL语句为：

```
UPDATE books SET price = price + 5 WHERE note = 'novel';
```

执行前先使用SELECT语句查看当前记录：

```
mysql> SELECT id, name, price, note FROM books WHERE note = 'novel';
+----+---------------------+-------+-------+
| id | name           | price | note  |
+----+---------------------+-------+-------+
|  1 | Tale of AAA    |    23 | novel |
|  3 | Story of Jane  |    40 | novel |
|  4 | Lovey Day      |    20 | novel |
+----+---------------+-------+-------+
3 rows in set (0.00 sec)
```

使用UPDATE语句执行更新操作：

```
mysql> UPDATE books SET price = price + 5 WHERE note = 'novel';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

由结果可以看到，该语句对3条记录进行了更新，使用SELECT语句查看更新结果：

```
mysql> SELECT id, name, price, note FROM books WHERE note = 'novel';
+----+-----------------+-------+-------+
| id | name         | price | note  |
+----+-----------------+---------+-------+
```

```
| 1 | Tale of AAA  |    28 | novel |
| 3 | Story of Jane |    45 | novel |
| 4 | Lovey Day   |    25 | novel |
+----+---------------+-----------+-------+
```

对比可知，price 的值都在原来的价格之上增加了 5。

步骤 04 将名称为 EmmaT 的书的价格改为 40，并将说明改为 drama。

修改语句为：

```
UPDATE books SET price=40,note= 'drama 'WHERE name= 'EmmaT ';
```

执行修改前，使用 SELECT 语句查看当前记录：

```
mysql> SELECT name, price, note FROM books WHERE name='EmmaT';
+---------+-------+------+
| name  | price | note |
+---------+-------+------+
| EmmaT |  35 | joke |
+---------+-------+------+
1 row in set (0.00 sec)
```

下面执行修改操作：

```
mysql> UPDATE books SET price=40,note='drama' WHERE name='EmmaT';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

结果显示修改了一条记录，使用 SELECT 查看执行结果：

```
mysql> SELECT name, price, note FROM books WHERE name='EmmaT';
+---------+--------+-------+
| name  | price | note  |
+---------+--------+-------+
| EmmaT |  40  | drama |
+---------+--------+-------+
1 row in set (0.00 sec)
```

可以看到，price 和 note 字段的值已经改变，修改操作成功。

步骤 05 删除库存为 0 的记录。

删除库存为 0 的语句为：

```
DELETE FROM books WHERE num=0;
```

删除之前使用 SELECT 语句查看当前记录：

```
mysql> SELECT * FROM books WHERE num=0;
+----+---------------+---------------+---------+---------+-------+------+
```

```
| id | name          | authors     | price | pubdate | note | num |
+----+---------------+-------------+-------+---------+------+-----+
|  3 | Story of Jane | Jane Tim    |    45 |  2001   | novel |   0 |
|  5 | Old Land      | Honore Blade |   30 |  2010   | law  |   0 |
+----+---------------+-------------+-------+---------+------+-----+
2 rows in set (0.00 sec)
```

可以看到，当前有两条记录的 num 值为 0，下面使用 DELETE 语句删除这两条记录，SQL 语句如下：

```
mysql> DELETE FROM books WHERE num=0;
Query OK, 2 rows affected (0.00 sec)
```

语句执行成功，查看操作结果：

```
mysql> SELECT * FROM books WHERE num=0;
Empty set (0.00 sec)
```

可以看到，查询结果为空，表中已经没有库存量为 0 的记录。

8.5 专家解惑

疑问 1：插入记录时可以不指定字段名称吗？

不管使用哪种 INSERT 语法，都必须给出 VALUES 的正确数目。如果不提供字段名，则必须给每个字段提供一个值，否则将产生一条错误消息。如果要在 INSERT 操作中省略某些字段，这些字段需要满足一定条件：该列定义为允许空值；或者表定义时给出默认值，如果不给出值，将使用默认值。

疑问 2：更新或者删除表时必须指定 WHERE 子句吗？

在前面章节中可以看到，所有的 UPDATE 和 DELETE 语句全都在 WHERE 子句中指定了条件。如果省略 WHERE 子句，则 UPDATE 或 DELETE 将被应用到表中所有的行。因此，除非确实打算更新或者删除所有记录，否则要注意使用不带 WHERE 子句的 UPDATE 或 DELETE 语句。建议在对表进行更新和删除操作之前，使用 SELECT 语句确认需要删除的记录，以免造成无法挽回的结果。

8.6 经典习题

创建数据表 pet，并对表进行插入、更新与删除操作，pet 表结构如表 8.3 所示。

（1）首先创建数据表 pet，使用不同的方法将表 8.4 中的记录插入到 pet 表中。

（2）使用 UPDATE 语句将名称为 Fang 的狗的主人改为 Kevin。

（3）将没有主人的宠物的 owner 字段值都改为 Duck。

（4）删除已经死亡的宠物记录。

（5）删除所有表中的记录。

表 8.3 pet 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
name	宠物名称	VARCHAR(20)	否	否	是	否	否
owner	宠物主人	VARCHAR(20)	否	否	否	否	否
species	种类	VARCHAR(20)	否	否	是	否	否
sex	性别	CHAR(1)	否	否	是	否	否
birth	出生日期	YEAR	否	否	是	否	否
death	死亡日期	YEAR	否	否	否	否	否

表 8.4 pet 表中记录

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	2003	2010
Claws	Gwen	cat	m	2004	NULL
Buffy	NULL	dog	f	2009	NULL
Fang	Benny	dog	m	2000	NULL
Bowser	Diane	dog	m	2003	2009
Chirpy	NULL	bird	f	2008	NULL