第 7 章

◀ 查询数据 ▶

学习目标Objective

数据库管理系统的一个最重要的功能就是数据查询，数据查询不应只是简单查询数据库中存储的数据，还应该根据需要对数据进行筛选，以及确定数据以什么样的格式显示。MySQL 提供了功能强大、灵活的语句来实现这些操作，本章将介绍如何使用 SELECT 语句查询数据表中的一列或多列数据、使用集合函数显示查询结果、连接查询、子查询以及使用正则表达式进行查询等。

内容导航Navigation

- 了解基本查询语句
- 掌握表单查询的方法
- 掌握如何使用几何函数查询
- 掌握连接查询的方法
- 掌握如何使用子查询
- 熟悉合并查询结果
- 熟悉如何为表和字段取别名
- 掌握如何使用正则表达式查询
- 掌握综合案例中数据表的查询操作技巧和方法

7.1 基本查询语句

MySQL 从数据表中查询数据的基本语句为 SELECT 语句。SELECT 语句的基本格式是：

```
SELECT
        {* | <字段列表>}
        [
            FROM <表1>,<表2>...
            [WHERE <表达式>
```

```
            [GROUP BY <group by definition>]
            [HAVING <expression> [{<operator> <expression>}...]]
            [ORDER BY <order by definition>]
            [LIMIT [<offset>,] <row count>]
        ]
SELECT [字段1,字段2,…,字段n]
FROM [表或视图]
WHERE [查询条件];
```

其中，各条子句的含义如下：

- {*|<字段列表>}包含星号通配符选择字段列表，表示查询的字段，其中字段列至少包含一个字段名称，如果要查询多个字段，多个字段之间用逗号隔开，最后一个字段后不要加逗号。
- FROM <表 1>,<表 2>...，表 1 和表 2 表示查询数据的来源，可以是单个或者多个。
- WHERE 子句是可选项，如果选择该项，将限定查询行必须满足的查询条件。
- GROUP BY <字段>，该子句告诉 MySQL 如何显示查询出来的数据，并按照指定的字段分组。
- [ORDER BY <字段 >]，该子句告诉 MySQL 按什么样的顺序显示查询出来的数据，可以进行的排序有：升序(ASC)、降序（DESC）。
- [LIMIT [<offset>,] <row count>]，该子句告诉 MySQL 每次显示查询出来的数据条数。

SELECT 的可选参数比较多，读者可能无法一下完全理解，不要紧，接下来将从最简单的开始，一步一步深入学习之后，读者会对各个参数的作用有清晰的认识。

下面以一个例子说明如何使用 SELECT 从单个表中获取数据。

首先定义数据表，输入语句如下：

```
CREATE TABLE fruits
(
f_id    char(10)      NOT NULL,
s_id    INT           NOT NULL,
f_name  char(255)     NOT NULL,
f_price decimal(8,2)      NOT NULL,
PRIMARY KEY(f_id)
);
```

为了演示如何使用 SELECT 语句，需要插入如下数据：

```
mysql> INSERT INTO fruits (f_id, s_id, f_name, f_price)
    -> VALUES('a1', 101,'apple',5.2),
    -> ('b1',101,'blackberry', 10.2),
    -> ('bs1',102,'orange', 11.2),
    -> ('bs2',105,'melon',8.2),
    -> ('t1',102,'banana', 10.3),
    -> ('t2',102,'grape', 5.3),
```

```
    -> ('o2',103,'coconut', 9.2),
    -> ('c0',101,'cherry', 3.2),
    -> ('a2',103, 'apricot',2.2),
    -> ('l2',104,'lemon', 6.4),
    -> ('b2',104,'berry', 7.6),
    -> ('m1',106,'mango', 15.7),
    -> ('m2',105,'xbabay', 2.6),
    -> ('t4',107,'xbababa', 3.6),
    -> ('m3',105,'xxtt', 11.6),
    -> ('b5',107,'xxxx', 3.6);
```

使用 SELECT 语句查询 f_id 和 f_name 字段的数据。

```
mysql> SELECT f_id, f_name FROM fruits;
+------+------------+
| f_id | f_name     |
+------+------------+
| a1   | apple      |
| a2   | apricot    |
| b1   | blackberry |
| b2   | berry      |
| b5   | xxxx       |
| bs1  | orange     |
| bs2  | melon      |
| c0   | cherry     |
| l2   | lemon      |
| m1   | mango      |
| m2   | xbabay     |
| m3   | xxtt       |
| o2   | coconut    |
| t1   | banana     |
| t2   | grape      |
| t4   | xbababa    |
+------+------------+
16 rows in set (0.00 sec)
```

该语句的执行过程是，SELECT 语句决定了要查询的列值，在这里查询 f_id 和 f_name 两个字段的值，FROM 子句指定了数据的来源，这里指定数据表 fruits，因此返回结果为 fruits 表中 f_id 和 f_name 两个字段下所有的数据。其显示顺序为添加到表中的顺序。

7.2 单表查询

单表查询是指从一张表数据中查询所需的数据。本节将介绍单表查询中的各种基本的查询

方式，主要有：查询所有字段、查询指定字段、查询指定记录、查询空值、多条件的查询、对查询结果进行排序等。

7.2.1 查询所有字段

1. 在 SELECT 语句中使用星号（*）通配符查询所有字段

SELECT 查询记录最简单的形式是从一个表中检索所有记录，实现的方法是使用星号（*）通配符指定查找所有列的名称。语法格式如下：

```
SELECT * FROM 表名;
```

【例 7.1】从 fruits 表中检索所有字段的数据，SQL 语句如下：

```
mysql> SELECT * FROM fruits;
+------+----------+---------------+-----------+
| f_id | s_id     | f_name        | f_price   |
+------+----------+---------------+-----------+
| a1   |  101     | apple         |    5.20   |
| a2   |  103     | apricot       |    2.20   |
| b1   |  101     | blackberry    |   10.20   |
| b2   |  104     | berry         |    7.60   |
| b5   |  107     | xxxx          |    3.60   |
| bs1  |  102     | orange        |   11.20   |
| bs2  |  105     | melon         |    8.20   |
| c0   |  101     | cherry        |    3.20   |
| l2   |  104     | lemon         |    6.40   |
| m1   |  106     | mango         |   15.70   |
| m2   |  105     | xbabay        |    2.60   |
| m3   |  105     | xxtt          |   11.60   |
| o2   |  103     | coconut       |    9.20   |
| t1   |  102     | banana        |   10.30   |
| t2   |  102     | grape         |    5.30   |
| t4   |  107     | xbababa       |    3.60   |
+------+----------+---------------+-----------+
```

可以看到，使用星号（*）通配符时，将返回所有列，列按照定义表时候的顺序显示。

2. 在 SELECT 语句中指定所有字段

下面介绍另外一种查询所有字段值的方法。根据前面 SELECT 语句的格式，SELECT 关键字后面的字段名为将要查找的数据，因此可以将表中所有字段的名称跟在 SELECT 子句后面，如果忘记了字段名称，可以使用 DESC 命令查看表的结构。有时候，由于表中的字段可能比较多，不一定能记得所有字段的名称，因此该方法会很不方便，不建议使用。例如查询 fruits 表中的所有数据，SQL 语句也可以书写如下，

```
SELECT f_id, s_id ,f_name, f_price FROM fruits;
```

查询结果与【例 7.1】相同。

一般情况下，除非需要使用表中所有的字段数据，最好不要使用通配符‘*’。使用通配符虽然可以节省输入查询语句的时间，但是获取不需要的列数据通常会降低查询和所使用的应用程序的效率。通配符的优势是，当不知道所需要的列的名称时，可以通过它获取它们。

7.2.2 查询指定字段

1. 查询单个字段

查询表中的某一个字段，语法格式为：

```
SELECT 列名 FROM 表名;
```

【例 7.2】查询 fruits 表中 f_name 列所有水果名称，SQL 语句如下：

```
SELECT f_name FROM fruits;
```

该语句使用 SELECT 声明从 fruits 表中获取名称为 f_name 字段下的所有水果名称，指定字段的名称紧跟在 SELECT 关键字之后，查询结果如下：

```
mysql> SELECT f_name FROM fruits;
+------------------+
| f_name           |
+------------------+
| apple            |
| apricot          |
| blackberry       |
| berry            |
| xxxx             |
| orange           |
| melon            |
| cherry           |
| lemon            |
| mango            |
| xbabay           |
| xxtt             |
| coconut          |
| banana           |
| grape            |
| xbababa          |
+------------------+
```

输出结果显示了 fruits 表中 f_name 字段下的所有数据。

2. 查询多个字段

使用 SELECT 声明，可以获取多个字段下的数据，只需要在关键字 SELECT 后面指定要查找的字段的名称，不同字段名称之间用逗号（，）分隔开，最后一个字段后面不需要加逗号，语法格式如下：

```
SELECT 字段名1,字段名2,…,字段名n  FROM 表名;
```

【例 7.3】例如，从 fruits 表中获取 f_name 和 f_price 两列，SQL 语句如下：

```
SELECT f_name, f_price FROM fruits;
```

该语句使用 SELECT 声明从 fruits 表中获取名称为 f_name 和 f_price 两个字段下的所有水果名称和价格，两个字段之间用逗号分隔开，查询结果如下：

```
mysql> SELECT f_name, f_price FROM fruits;
+--------------+-----------+
| f_name       | f_price   |
+--------------+-----------+
| apple        |    5.20   |
| apricot      |    2.20   |
| blackberry   |   10.20   |
| berry        |    7.60   |
| xxxx         |    3.60   |
| orange       |   11.20   |
| melon        |    8.20   |
| cherry       |    3.20   |
| lemon        |    6.40   |
| mango        |   15.70   |
| xbabay       |    2.60   |
| xxtt         |   11.60   |
| coconut      |    9.20   |
| banana       |   10.30   |
| grape        |    5.30   |
| xbababa      |    3.60   |
+--------------+-----------+
```

输出结果显示了 fruits 表中 f_name 和 f_price 两个字段下的所有数据。

MySQL 中的 SQL 语句是不区分大小写的，因此 SELECT 和 select 作用是相同的，但是，许多开发人员习惯将关键字使用大写，而数据列和表名使用小写，读者也应该养成一个良好的编程习惯，这样写出来的代码更容易阅读和维护。

7.2.3 查询指定记录

数据库中包含大量的数据，根据特殊要求，可能只需要查询表中的指定数据，即对数据进行过滤。在 SELECT 语句中，通过 WHERE 子句可以对数据进行过滤，语法格式为：

```
SELECT 字段名1,字段名2,…,字段名 n
FROM 表名
WHERE 查询条件
```

在 WHERE 子句中，MySQL 提供了一系列的条件判断符，查询结果如表 7.1 所示。

表 7.1 WHERE 条件判断符

操作符	说明
=	相等
<> , !=	不相等
<	小于
<=	小于或者等于
>	大于
>=	大于或者等于
BETWEEN	位于两值之间

【例 7.4】查询价格为 10.2 元的水果的名称，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_price = 10.2;
```

该语句使用 SELECT 声明从 fruits 表中获取价格等于 10.2 的水果的数据，从查询结果可以看到，价格是 10.2 的水果的名称是 blackberry，其他的均不满足查询条件，查询结果如下

```
mysql> SELECT f_name, f_price
    -> FROM fruits
    -> WHERE f_price = 10.2;
+------------+---------+
| f_name     | f_price |
+------------+---------+
| blackberry | 10.20   |
+------------+---------+
```

本例采用了简单的相等过滤，查询一个指定列 f_price 具有值 10.20。

相等还可以用来比较字符串，如下：

【例 7.5】查找名称为“apple”的水果的价格，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_name = 'apple';
```

该语句使用 SELECT 声明从 fruits 表中获取名称为“apple”的水果的价格，从查询结果可以看到只有名称为“apple”行被返回，其他的均不满足查询条件。

```
mysql> SELECT f_name, f_price
    -> FROM fruits
    -> WHERE f_name = 'apple';
+--------+---------+
| f_name | f_price |
+--------+---------+
| apple  | 5.20    |
+--------+---------+
```

【例 7.6】查询价格小于 10 的水果的名称，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_price < 10;
```

该语句使用 SELECT 声明从 fruits 表中获取价格低于 10 的水果名称，即 f_price 小于 10 的水果信息被返回，查询结果如下：

```
mysql> SELECT f_name, f_price
    -> FROM fruits
    -> WHERE f_price < 10.00;
+-----------+-----------+
| f_name    | f_price   |
+-----------+-----------+
| apple     |      5.20 |
| apricot   |      2.20 |
| berry     |      7.60 |
| xxxx      |      3.60 |
| melon     |      8.20 |
| cherry    |      3.20 |
| lemon     |      6.40 |
| xbabay    |      2.60 |
| coconut   |      9.20 |
| grape     |      5.30 |
| xbababa   |      3.60 |
+-----------+-----------+
```

可以看到查询结果中，所有记录的 f_price 字段的值均小于 10.00 元。而大于或等于 10.00 元的记录没有被返回。

7.2.4 带 IN 关键字的查询

IN 操作符用来查询满足指定范围内的条件的记录，使用 IN 操作符，将所有检索条件用括

号括起来，检索条件之间用逗号分隔开，只要满足条件范围内的一个值即为匹配项。

【例 7.7】查询 s_id 为 101 和 102 的记录，SQL 语句如下：

```
SELECT s_id,f_name, f_price
FROM fruits
WHERE s_id IN (101,102)
ORDER BY f_name;
```

查询结果如下：

```
+------+------------+-------------+
| s_id | f_name     | f_price     |
+------+------------+-------------+
|  101 | apple      |        5.20 |
|  102 | banana     |       10.30 |
|  101 | blackberry |       10.20 |
|  101 | cherry     |        3.20 |
|  102 | grape      |        5.30 |
|  102 | orange     |       11.20 |
+------+------------+-------------+
```

相反的，可以使用关键字 NOT 来检索不在条件范围内的记录。

【例 7.8】查询所有 s_id 不等于 101 也不等于 102 的记录，SQL 语句如下：

```
SELECT s_id,f_name, f_price
FROM fruits
WHERE s_id NOT IN (101,102)
ORDER BY f_name;
```

查询结果如下：

```
+------+---------+-------------+
| s_id | f_name  | f_price     |
+------+---------+-------------+
|  103 | apricot |        2.20 |
|  104 | berry   |        7.60 |
|  103 | coconut |        9.20 |
|  104 | lemon   |        6.40 |
|  106 | mango   |       15.70 |
|  105 | melon   |        8.20 |
|  107 | xbababa |        3.60 |
|  105 | xbabay  |        2.60 |
|  105 | xxtt    |       11.60 |
|  107 | xxxx    |        3.60 |
+------+---------+-------------+
```

可以看到，该语句在 IN 关键字前面加上了 NOT 关键字，这使得查询的结果与前面一个

的结果正好相反，前面检索了 s_id 等于 101 和 102 的记录，而这里所要求的查询的记录中的 s_id 字段值不等于这两个值中的任何一个。

7.2.5 带 BETWEEN AND 的范围查询

BETWEEN AND 用来查询某个范围内的值，该操作符需要两个参数，即范围的开始值和结束值，如果字段值满足指定的范围查询条件，则这些记录被返回。

【例 7.9】查询价格在 2.00 元到 10.20 元之间的水果名称和价格，SQL 语句如下：

```
SELECT f_name, f_price FROM fruits WHERE f_price BETWEEN 2.00 AND 10.20;
```

查询结果如下：

```
mysql> SELECT f_name, f_price
    -> FROM fruits
    -> WHERE f_price BETWEEN 2.00 AND 10.20;
+--------------+-----------+
| f_name       | f_price   |
+--------------+-----------+
| apple        |      5.20 |
| apricot      |      2.20 |
| blackberry   |     10.20 |
| berry        |      7.60 |
| xxxx         |      3.60 |
| melon        |      8.20 |
| cherry       |      3.20 |
| lemon        |      6.40 |
| xbabay       |      2.60 |
| coconut      |      9.20 |
| grape        |      5.30 |
| xbababa      |      3.60 |
+--------------+-----------+
```

可以看到，返回结果包含了价格从 2.00 元到 10.20 元之间的字段值，并且端点值 10.20 也包括在返回结果中，即 BETWEEN 匹配范围中所有值，包括开始值和结束值。

BETWEEN AND 操作符前可以加关键字 NOT，表示指定范围之外的值，如果字段值不满足指定的范围内的值，则这些记录被返回。

【例 7.10】查询价格在 2.00 元到 10.20 元之外的水果名称和价格，SQL 语句如下：

```
SELECT f_name, f_price
FROM fruits
WHERE f_price NOT BETWEEN 2.00 AND 10.20;
```

查询结果如下：

```
+--------+----------+
| f_name | f_price |
+--------+----------+
| orange |   11.20 |
| mango |   15.70 |
| xxtt   |   11.60 |
| banana |   10.30 |
+--------+----------+
```

由结果可以看到，返回的记录只有 f_price 字段大于 10.20 的，其实，f_price 字段小于 2.0(的记录也满足查询条件。因此，如果表中有 f_price 字段小于 2.00 的记录，也应当作为查询结果。

7.2.6 带 LIKE 的字符匹配查询

在前面的检索操作中，讲述了如何查询多个字段的记录，如何进行比较查询或者是查询一个条件范围内的记录，如果要查找所有的包含字符“ge”的水果名称，该如何查找呢？简单的比较操作在这里已经行不通了，在这里，需要使用通配符进行匹配查找，通过创建查找模式对表中的数据进行比较。执行这个任务的关键字是 LIKE。

通配符是一种在 SQL 的 WHERE 条件子句中拥有特殊意思的字符，SQL 语句中支持多种通配符，可以和 LIKE 一起使用的通配符有‘%’和‘_’。

1. 百分号通配符‘%’，匹配任意长度的字符，甚至包括零字符

【例 7.11】查找所有以‘b’字母开头的水果，SQL 语句如下：

```
SELECT f_id, f_name
FROM fruits
WHERE f_name LIKE 'b%';
```

查询结果如下：

```
+-------+-------------+
| f_id  | f_name      |
+-------+-------------+
| b1    | blackberry  |
| b2    | berry       |
| t1    | banana      |
+-------+-------------+
```

该语句查询的结果返回所有以‘b’开头的水果的 id 和 name，‘%’告诉 MySQL，返回所有以字母‘b’开头的记录，不管‘b’后面有多少个字符。

在搜索匹配时通配符‘%’可以放在不同位置。如【例 7.12】。

【例 7.12】在 fruits 表中，查询 f_name 中包含字母‘g’的记录，SQL 语句如下：

```
SELECT f_id, f_name
FROM fruits
WHERE f_name LIKE '%g%';
```

查询结果如下：

```
+------+--------+
| f_id | f_name |
+------+--------+
| bs1  | orange |
| m1   | mango  |
| t2   | grape  |
+------+--------+
```

该语句查询字符串中包含字母‘g’的水果名称，只要名字中有字符‘g’，而前面或后面不管有多少个字符，都满足查询的条件。

【例 7.13】查询以‘b’开头，并以‘y’结尾的水果的名称，SQL 语句如下：

```
SELECT f_name
FROM fruits
WHERE f_name LIKE 'b%y';
```

查询结果如下：

```
+-------------+
| f_name      |
+-------------+
| blackberry  |
| berry       |
+-------------+
```

通过以上查询结果，可以看到，‘%’用于匹配在指定的位置的任意数目的字符。

2. 下划线通配符'_'，一次只能匹配任意一个字符

另一个非常有用的通配符是下划线通配符‘_’，该通配符的用法和‘%’相同，区别是‘%’可以匹配多个字符，而‘_’只能匹配任意单个字符，如果要匹配多个字符，则需要使用相同个数的‘_’。

【例 7.14】在 fruits 表中，查询以字母‘y’结尾，且‘y’前面只有 4 个字母的记录，SQL 语句如下：

```
SELECT f_id, f_name FROM fruits WHERE f_name LIKE '____y';
```

查询结果如下：

```
+------+--------+
| f_id | f_name |
```

```
+------+--------+
| b2   | berry  |
+------+--------+
```

从结果可以看到，以‘y’结尾且前面只有 4 个字母的记录只有一条。其他记录的 f_nam 字段也有以‘y’结尾的，但其总的字符串长度不为 5，因此不在返回结果中。

7.2.7 查询空值

数据表创建的时候，设计者可以指定某列中是否可以包含空值（NULL）。空值不同于也不同于空字符串。空值一般表示数据未知、不适用或将在以后添加数据。在 SELECT 语中使用 IS NULL 子句，可以查询某字段内容为空的记录。

下面，在数据库中创建数据表 customers，该表中包含了本章中需要用到的数据。

```
CREATE TABLE customers
(
  c_id      int       NOT NULL AUTO_INCREMENT,
  c_name    char(50)  NOT NULL,
  c_address char(50)  NULL,
  c_city    char(50)  NULL,
  c_zip     char(10)  NULL,
  c_contact char(50)  NULL,
  c_email   char(255) NULL,
  PRIMARY KEY (c_id)
);
```

为了演示需要插入数据，请读者执行以下语句。

```
INSERT INTO customers(c_id, c_name, c_address, c_city,
c_zip,  c_contact, c_email)
VALUES(10001, 'RedHook', '200 Street ', 'Tianjin',
 '300000',  'LiMing', 'LMing@163.com'),
(10002, 'Stars', '333 Fromage Lane',
 'Dalian', '116000',  'Zhangbo','Jerry@hotmail.com'),
(10003, 'Netbhood', '1 Sunny Place', 'Qingdao',  '266000',
 'LuoCong', NULL),
(10004, 'JOTO', '829 Riverside Drive', 'Haikou',
 '570000',  'YangShan', 'sam@hotmail.com');
 SELECT COUNT(*) AS cust_num  FROM customers;
```

【例 7.15】查询 customers 表中 c_email 为空的记录的 c_id、c_name 和 c_email 字段值，SQ 语句如下：

```
SELECT c_id, c_name,c_email FROM customers WHERE c_email IS NULL;
```

查询结果如下：

```
mysql> SELECT c_id, c_name,c_email FROM customers WHERE c_email IS NULL;
+-------+----------+---------+
| c_id  | c_name | c_email |
+-------+----------+---------+
| 10003 | Netbhood | NULL|
+-------+----------+---------+
```

可以看到，显示 customers 表中字段 c_email 的值为 NULL 的记录，满足查询条件。

与 IS NULL 相反的是 NOT IS NULL，该关键字查找字段不为空的记录。

【例 7.16】查询 customers 表中 c_email 不为空的记录的 c_id、c_name 和 c_email 字段值，SQL 语句如下：

```
SELECT c_id, c_name,c_email FROM customers WHERE c_email IS NOT NULL;
```

查询结果如下：

```
mysql> SELECT c_id, c_name,c_email FROM customers WHERE c_email IS NOT NULL;
+-------+------------+----------------------+
| c_id  | c_name  | c_email        |
+-------+------------+----------------------+
| 10001 | RedHook | LMing@163.com  |
| 10002 | Stars   | Jerry@hotmail.com |
| 10004 | JOTO    | sam@hotmail.com  |
+-------+------------+----------------------+
```

可以看到，查询出来的记录的 c_email 字段都不为空值。

7.2.8 带 AND 的多条件查询

使用 SELECT 查询时，可以增加查询的限制条件，这样可以使查询的结果更加精确。MySQL 在 WHERE 子句中使用 AND 操作符限定只有满足所有查询条件的记录才会被返回。可以使用 AND 连接两个甚至多个查询条件，多个条件表达式之间用 AND 分开。

【例 7.17】在 fruits 表中查询 s_id = 101，并且 f_price 大于等于 5 的水果价格和名称，SQL 语句如下：

```
SELECT f_id, f_price, f_name FROM fruits WHERE s_id = '101' AND f_price >=5;
```

查询结果如下：

```
mysql> SELECT f_id, f_price, f_name
    -> FROM fruits
    -> WHERE s_id = '101' AND f_price >= 5;
+------+---------+------------+
| f_id | f_price | f_name   |
+------+---------+------------+
| a1   | 5.20 | apple    |
```

```
| b1   | 10.20 | blackberry |
+------+-------+------------+
```

前面的语句检索了 s_id=101 的水果供应商所有价格大于等于 5 元的水果名称和价格。WHERE 子句中的条件分为两部分，AND 关键字指示 MySQL 返回所有同时满足两个条件的行。即使是 id=101 的水果供应商提供的水果，如果价格<5，或者是 id 不等于‘101’的水果供应商里的水果不管其价格为多少，均不是要查询的结果。

上述例子的 WHERE 子句中只包含了一个 AND 语句，把两个过滤条件组合在一起。实际上可以添加多个 AND 过滤条件，增加条件的同时增加一个 AND 关键字。

【例 7.18】在 fruits 表中查询 s_id = 101 或者 102，且 f_price 大于 5，并且 f_name='apple' 的水果价格和名称，SQL 语句如下：

```
SELECT f_id, f_price, f_name FROM fruits
WHERE s_id IN('101', '102') AND f_price >= 5 AND f_name = 'apple';
```

查询结果如下：

```
mysql> SELECT f_id, f_price, f_name FROM fruits
    -> WHERE s_id IN('101','102') AND f_price >= 5 AND f_name = 'apple';
+------+---------+--------+
| f_id | f_price | f_name |
+------+---------+--------+
| a1   |    5.20 | apple  |
+------+---------+--------+
```

可以看到，符合查询条件的返回记录只有一条。

7.2.9 带 OR 的多条件查询

与 AND 相反，在 WHERE 声明中使用 OR 操作符，表示只需要满足其中一个条件的记录即可返回。OR 也可以连接两个甚至多个查询条件，多个条件表达式之间用 OR 分开。

【例 7.19】查询 s_id=101 或者 s_id=102 的水果供应商的 f_price 和 f_name，SQL 语句如下：

```
SELECT s_id,f_name, f_price FROM fruits WHERE s_id = 101 OR s_id = 102;
```

查询结果如下：

```
mysql> SELECT s_id,f_name, f_price
    -> FROM fruits
    -> WHERE s_id = 101 OR s_id = 102;
+------+--------------+-----------+
| s_id | f_name       | f_price   |
+------+--------------+-----------+
|  101 | apple        |      5.20 |
```

```
|  101 | blackberry      |   10.20  |
|  102 | orange          |   11.20  |
|  101 | cherry          |    3.20  |
|  102 | banana          |   10.30  |
|  102 | grape           |    5.30  |
+------+-----------------+----------+
```

结果显示了 s_id=101 和 s_id=102 的商店里的水果名称和价格，OR 操作符告诉 MySQL，检索的时候只需要满足其中的一个条件，不需要全部都满足。如果这里使用 AND 的话，将检索不到符合条件的数据。

在这里，也可以使用 IN 操作符实现与 OR 相同的功能，下面的例子可进行说明。

【例 7.20】查询 s_id=101 或者 s_id=102 的水果供应商的 f_price 和 f_name，SQL 语句如下：

```
SELECT s_id,f_name, f_price FROM fruits WHERE s_id IN(101,102);
```

查询结果如下：

```
mysql> SELECT s_id,f_name, f_price
    -> FROM fruits
    -> WHERE s_id IN(101,102);
+------+-----------------+----------+
| s_id | f_name      | f_price |
+------+-----------------+----------+
|  101 | apple       |    5.20  |
|  101 | blackberry  |   10.20  |
|  102 | orange      |   11.20  |
|  101 | cherry      |    3.20  |
|  102 | banana      |   10.30  |
|  102 | grape       |    5.30  |
+------+-----------------+----------+
```

在这里可以看到，OR 操作符和 IN 操作符使用后的结果是一样的，它们可以实现相同的功能。但是使用 IN 操作符使得检索语句更加简洁明了，并且 IN 执行的速度要快于 OR。更重要的是，使用 IN 操作符，可以执行更加复杂的嵌套查询（后面章节将会讲述）。

OR 可以和 AND 一起使用，但是在使用时要注意两者的优先级，由于 AND 的优先级高于 OR，因此先对 AND 两边的操作数进行操作，再与 OR 中的操作数结合。

7.2.10 查询结果不重复

从前面的例子可以看到，SELECT 查询返回所有匹配的行。例如，查询 fruits 表中所有的 s_id，其结果为：

```
+------+
| s_id |
```

```
+------+
| 101 |
| 103 |
| 101 |
| 104 |
| 107 |
| 102 |
| 105 |
| 101 |
| 104 |
| 106 |
| 105 |
| 105 |
| 103 |
| 102 |
| 102 |
| 107 |
+------+
```

可以看到查询结果返回了 16 条记录，其中有一些重复的 s_id 值，有时，出于对数据的要求，需要消除重复的记录值，如何使查询结果没有重复呢？在 SELECT 语句中，可用 DISTINCT 关键字指示 MySQL 消除重复的记录值。语法格式为：

```
SELECT DISTINCT 字段名 FROM 表名;
```

【例 7.21】查询 fruits 表中 s_id 字段的值，返回 s_id 字段值且不得重复，SQL 语句如

```
SELECT DISTINCT s_id FROM fruits;
```

查询结果如下：

```
mysql> SELECT DISTINCT s_id FROM fruits;
+------+
| s_id |
+------+
| 101 |
| 103 |
| 104 |
| 107 |
| 102 |
| 105 |
| 106 |
+------+
```

可以看到，这次查询结果只返回了 7 条记录的 s_id 值，且不再有重复的值，SELE
DISTINCT s_id 告诉 MySQL 只返回不同的 s_id 行。

7.2.11 对查询结果排序

从前面的查询结果，读者会发现有些字段的值是没有任何顺序的，MySQL 可以通过在 SELECT 语句中使用 ORDER BY 子句，对查询的结果进行排序。

1. 单列排序

例如查询 f_name 字段，查询结果如下：

```
mysql> SELECT f_name FROM fruits;
+--------------+
| f_name       |
+--------------+
| apple        |
| apricot      |
| blackberry   |
| berry        |
| xxxx         |
| orange       |
| melon        |
| cherry       |
| lemon        |
| mango        |
| xbabay       |
| xxtt         |
| coconut      |
| banana       |
| grape        |
| xbababa      |
+--------------+
```

可以看到，查询的数据并没有以一种特定的顺序显示，如果没有对它们进行排序，它们将根据它们插入到数据表中的顺序来显示。

下面使用 ORDER BY 子句对指定的列数据进行排序。

【例 7.22】查询 fruits 表的 f_name 字段值，并对其进行排序，SQL 语句如下：

```
mysql> SELECT f_name FROM fruits ORDER BY f_name;
+--------------+
| f_name       |
+--------------+
| apple        |
| apricot      |
| banana       |
| berry        |
| blackberry   |
| cherry       |
```

```
| coconut     |
| grape       |
| lemon       |
| mango       |
| melon       |
| orange      |
| xbababa     |
| xbabay      |
| xxtt        |
| xxxx        |
+-------------------+
```

该语句查询的结果和前面的语句相同，不同的是，通过指定 ORDER BY 子句，MySQ
对查询的 name 列的数据，按字母表的顺序进行了升序排序。

2. 多列排序

有时，需要根据多列值进行排序。比如，如果要显示一个学生列表，可能会有多个学生
姓氏是相同的，因此还需要根据学生的名进行排序。对多列数据进行排序，须将需要排序的
之间用逗号隔开。

【例 7.23】查询 fruits 表中的 f_name 和 f_price 字段，先按 f_name 排序，再按 f_price 排序
SQL 语句如下：

```
SELECT f_name, f_price FROM fruits ORDER BY f_name, f_price;
```

查询结果如下：

```
mysql> SELECT f_name, f_price FROM fruits ORDER BY f_name, f_price;
+---------------+-----------+
| f_name    | f_price |
+---------------+-------------+
| apple      |    5.20  |
| apricot    |    2.20  |
| banana    |   10.30  |
| berry      |    7.60  |
| blackberry  |   10.20  |
| cherry     |    3.20  |
| coconut    |    9.20  |
| grape      |    5.30  |
| lemon      |    6.40  |
| mango     |   15.70  |
| melon      |    8.20  |
| orange     |   11.20  |
| xbababa    |    3.60  |
| xbabay     |    2.60  |
| xxtt       |   11.60  |
| xxxx       |    3.60  |
```

```
+----------------+-------------+
```

在对多列进行排序的时候，首先排序的第一列必须有相同的列值，才会对第二列进行排序。如果第一列数据中所有值都是唯一的，将不再对第二列进行排序。

3. 指定排序方向

默认情况下，查询数据按字母升序进行排序（从 A~Z），但数据的排序并不仅限于此，还可以使用 ORDER BY 对查询结果进行降序排序（从 Z~A），这可以通过关键字 DESC 实现，下面的例子表明了如何进行降序排列。

【例 7.24】查询 fruits 表中的 f_name 和 f_price 字段，对结果按 f_price 降序方式排序，SQL 语句如下：

```
SELECT f_name, f_price FROM fruits ORDER BY f_price DESC;
```

查询结果如下：

```
mysql> SELECT f_name,  f_price FROM fruits ORDER BY f_price DESC;
+--------------+---------+
| f_name       | f_price |
+--------------+---------+
| mango        |   15.70 |
| xxtt         |   11.60 |
| orange       |   11.20 |
| banana       |   10.30 |
| blackberry   |   10.20 |
| coconut      |    9.20 |
| melon        |    8.20 |
| berry        |    7.60 |
| lemon        |    6.40 |
| grape        |    5.30 |
| apple        |    5.20 |
| xxxx         |    3.60 |
| xbababa      |    3.60 |
| cherry       |    3.20 |
| xbabay       |    2.60 |
| apricot      |    2.20 |
+--------------+---------+
```

与 DESC 相反的是 ASC（升序排序），将字段列中的数据，按字母表顺序升序排序。实际上，在排序的时候 ASC 是作为默认的排序方式，所以加不加都可以。

也可以对多列进行不同的顺序排序，如【例 7.25】所示。

【例 7.25】查询 fruits 表，先按 f_price 降序排序，再按 f_name 字段升序排序，SQL 语句如下：

```
SELECT f_price, f_name FROM fruits ORDER BY f_price DESC, f_name;
```

查询结果如下：

```
mysql> SELECT f_price, f_name FROM fruits ORDER BY f_price DESC, f_name;
+-----------+------------+
| f_price  | f_name   |
+-----------+------------+
|   15.70  | mango   |
|   11.60  | xxtt    |
|   11.20  | orange   |
|   10.30  | banana   |
|   10.20  | blackberry |
|    9.20  | coconut    |
|    8.20  | melon     |
|    7.60  | berry     |
|    6.40  | lemon     |
|    5.30  | grape     |
|    5.20  | apple     |
|    3.60  | xbababa    |
|    3.60  | xxxx      |
|    3.20  | cherry    |
|    2.60  | xbabay    |
|    2.20  | apricot    |
+-----------+------------+
```

DESC 排序方式只应用到直接位于其前面的字段上，由结果可以看出。

DESC 关键字只对其前面的列进行降序排列，在这里只对 f_price 排序，而并没有对 f_name 进行排序，因此，f_price 按降序排序，而 f_name 列仍按升序排序。如果要对多列都进行降序排序，必须要在每一列的列名后面加 DESC 关键字。

7.2.12 分组查询

分组查询是对数据按照某个或多个字段进行分组，MySQL 中使用 GROUP BY 关键字对数据进行分组，基本语法形式为：

```
[GROUP BY 字段] [HAVING <条件表达式>]
```

字段值为进行分组时所依据的列名称；“HAVING <条件表达式>”指定满足表达式限定条件的结果将被显示。

1. 创建分组

GROUP BY 关键字通常和集合函数一起使用，例如：MAX()、MIN()、COUNT()、SUM()、AVG()。例如，要返回每个水果供应商提供的水果种类，这时就要在分组过程中用到 COUNT() 函数，把数据分为多个逻辑组，并对每个组进行集合计算。

【例 7.26】根据 s_id 对 fruits 表中的数据进行分组，SQL 语句如下：

```
SELECT s_id, COUNT(*) AS Total FROM fruits GROUP BY s_id;
```

查询结果如下：

```
mysql> SELECT s_id, COUNT(*) AS Total FROM fruits GROUP BY s_id;
+------+------------+
| s_id | Total      |
+------+------------+
|  101 |     3      |
|  102 |     3      |
|  103 |     2      |
|  104 |     2      |
|  105 |     3      |
|  106 |     1      |
|  107 |     2      |
+------+------------+
```

查询结果显示，s_id 表示供应商的 ID，Total 字段使用 COUNT()函数计算得出，GROUP BY 子句按照 s_id 排序并对数据分组，可以看到 ID 为 101、102、105 的供应商分别提供 3 种水果，ID 为 103、104、107 的供应商分别提供 2 种水果，ID 为 106 的供应商只提供 1 种水果。

如果要查看每个供应商提供的水果的种类的名称，该怎么办呢？MySQL 中可以在 GROUP BY 字节中使用 GROUP_CONCAT()函数，将每个分组中各个字段的值显示出来。

【例 7.27】根据 s_id 对 fruits 表中的数据进行分组，将每个供应商的水果名称显示出来，SQL 语句如下：

```
SELECT s_id, GROUP_CONCAT(f_name) AS Names FROM fruits GROUP BY s_id;
```

查询结果如下：

```
mysql> SELECT s_id, GROUP_CONCAT(f_name) AS Names FROM fruits GROUP BY s_id;
+------+------------------------------+
| s_id | Names                        |
+------+------------------------------+
|  101 | apple,blackberry,cherry      |
|  102 | grape,banana,orange          |
|  103 | apricot,coconut              |
|  104 | lemon,berry                  |
|  105 | xbabay,xxtt,melon            |
```

```
| 106 | mango              |
| 107 | xxxx,xbababa       |
+------+--------------------+
```

由结果可以看到，GROUP_CONCAT()函数将每个分组中的名称显示出来了，其名称的个数与 COUNT()函数计算出来的相同。

2. 使用 HAVING 过滤分组

GROUP BY 可以和 HAVING 一起限定显示记录所需满足的条件，只有满足条件的分组会被显示。

【例 7.28】根据 s_id 对 fruits 表中的数据进行分组，并显示水果种类大于 1 的分组信息，SQL 语句如下：

```
SELECT s_id, GROUP_CONCAT(f_name) AS Names
FROM fruits
GROUP BY s_id HAVING COUNT(f_name) > 1;
```

查询结果如下：

```
+------+---------------------------------+
| s_id | Names                           |
+------+---------------------------------+
| 101 | apple,blackberry,cherry  |
| 102 | grape,banana,orange      |
| 103 | apricot,coconut          |
| 104 | lemon,berry              |
| 105 | xbabay,xxtt,melon        |
| 107 | xxxx,xbababa             |
+------+---------------------------------+
```

由结果可以看到，ID 为 101、102、103、104、105、107 的供应商提供的水果种类大于 1，满足 HAVING 子句条件，因此出现在返回结果中；而 ID 为 106 的供应商的水果种类等于 1，不满足限定条件，因此不在返回结果中。

HAVING 关键字与 WHERE 关键字都是用来过滤数据，两者有什么区别呢？其中重要的一点是，HAVING 在数据分组之后进行过滤来选择分组，而 WHERE 在分组之前用来选择记录。另外 WHERE 排除的记录不再包括在分组中。

3. 在 GROUP BY 子句中使用 WITH ROLLUP

使用 WITH ROLLUP 关键字之后，在所有查询出的分组记录之后增加一条记录，该记录计算查询出的所有记录的总和，即统计记录数量。

【例 7.29】根据 s_id 对 fruits 表中的数据进行分组，并显示记录数量，SQL 语句如下：

```
SELECT s_id, COUNT(*) AS Total
FROM fruits
GROUP BY s_id WITH ROLLUP;
```

查询结果如下：

```
+------+----------+
| s_id | Total    |
+------+----------+
|  101 |        3 |
|  102 |        3 |
|  103 |        2 |
|  104 |        2 |
|  105 |        3 |
|  106 |        1 |
|  107 |        2 |
| NULL |       16 |
+------+----------+
```

由结果可以看到，通过 GROUP BY 分组之后，在显示结果的最后面新添加了一行，该行 Total 列的值正好是上面所有数值之和。

4. 多字段分组

使用 GROUP BY 可以对多个字段进行分组，GROUP BY 关键字后面跟需要分组的字段，MySQL 根据多字段的值来进行层次分组，分组层次从左到右，即先按第 1 个字段分组，然后在第 1 个字段值相同的记录中，再根据第 2 个字段的值进行分组...依次类推。

【例 7.30】根据 s_id 和 f_name 字段对 fruits 表中的数据进行分组， SQL 语句如下，

```
mysql> SELECT * FROM fruits group by s_id,f_name;
```

查询结果如下：

```
+------+------+-----------+---------+
| f_id | s_id | f_name    | f_price |
+------+------+-----------+---------+
| a1   |  101 | apple     |    5.20 |
| b1   |  101 | blackberry|   10.20 |
| c0   |  101 | cherry    |    3.20 |
| t1   |  102 | banana    |   10.30 |
| t2   |  102 | grape     |    5.30 |
| bs1  |  102 | orange    |   11.20 |
| a2   |  103 | apricot   |    2.20 |
| c2   |  103 | coconut   |    9.20 |
| b2   |  104 | berry     |    7.60 |
| l2   |  104 | lemon     |    6.40 |
| bs2  |  105 | melon     |    8.20 |
```

```
| m2   |  105  | xbabay     |   2.60  |
| m3   |  105  | xxtt       |  11.60  |
| m1   |  106  | mango      |  15.70  |
| t4   |  107  | xbababa    |   3.60  |
| b5   |  107  | xxxx       |   3.60  |
+------+-------+------------+---------+
```

由结果可以看到，查询记录先按照 s_id 进行分组，再对 f_name 字段按不同的取值进行分组。

5. GROUP BY 和 ORDER BY 一起使用

某些情况下需要对分组进行排序，在前面的介绍中，ORDER BY 用来对查询的记录排序，如果和 GROUP BY 一起使用可以完成对分组的排序。

为了演示效果，首先创建数据表，SQL 语句如下：

```
CREATE TABLE orderitems
(
  o_num      int          NOT NULL,
  o_item     int          NOT NULL,
  f_id       char(10)     NOT NULL,
  quantity   int          NOT NULL,
  item_price decimal(8,2) NOT NULL,
  PRIMARY KEY (o_num,o_item)
) ;
```

然后插入演示数据。SQL 语句如下：

```
INSERT INTO orderitems(o_num, o_item, f_id, quantity, item_price)
VALUES(30001, 1, 'a1', 10, 5.2),
(30001, 2, 'b2', 3, 7.6),
(30001, 3, 'bs1', 5, 11.2),
(30001, 4, 'bs2', 15, 9.2),
(30002, 1, 'b3', 2, 20.0),
(30003, 1, 'c0', 100, 10),
(30004, 1, 'o2', 50, 2.50),
(30005, 1, 'c0', 5, 10),
(30005, 2, 'b1', 10, 8.99),
(30005, 3, 'a2', 10, 2.2),
(30005, 4, 'm1', 5, 14.99);
```

【例 7.31】查询订单价格大于 100 的订单号和总订单价格，SQL 语句如下：

```
SELECT o_num,  SUM(quantity * item_price) AS orderTotal
FROM orderitems
GROUP BY o_num
HAVING SUM(quantity*item_price) >= 100;
```

查询结果如下：

```
+-------+------------+
| o_num | orderTotal |
+-------+------------+
| 30001 |  268.80    |
| 30003 | 1000.00    |
| 30004 |  125.00    |
| 30005 |  236.85    |
+-------+------------+
```

可以看到，返回的结果中 orderTotal 列的总订单价格并没有按照一定顺序显示，接下来，使用 ORDER BY 关键字按总订单价格排序显示结果，SQL 语句如下：

```
SELECT o_num, SUM(quantity * item_price) AS orderTotal
FROM orderitems
GROUP BY o_num
HAVING SUM(quantity*item_price) >= 100
ORDER BY orderTotal;
```

查询结果如下：

```
+---------+--------------+
| o_num | orderTotal   |
+---------+--------------+
| 30004 |     125.00   |
| 30005 |     236.85   |
| 30001 |     268.80   |
| 30003 |    1000.00   |
+---------+--------------+
```

由结果可以看到，GROUP BY 子句按订单号对数据进行分组，SUM()函数便可以返回总的订单价格，HAVING 子句对分组数据进行过滤，使得只返回总价格大于 100 的订单，最后使用 ORDER BY 子句排序输出。

当使用 ROLLUP 时，不能同时使用 ORDER BY 子句进行结果排序。即 ROLLUP 和 ORDER BY 是互相排斥的。

7.2.13 使用 LIMIT 限制查询结果的数量

SELECT 返回所有匹配的行，有可能是表中所有的行，如仅仅需要返回第一行或者前几行，使用 LIMIT 关键字，基本语法格式如下：

```
LIMIT [位置偏移量,] 行数
```

第一个“位置偏移量”参数指示 MySQL 从哪一行开始显示，是一个可选参数，如果不指

定“位置偏移量”，将会从表中的第一条记录开始（第一条记录的位置偏移量是 0，第二条记录的位置偏移量是 1...依次类推）；第二个参数“行数”指示返回的记录条数。

【例 7.32】显示 fruits 表查询结果的前 4 行，SQL 语句如下：

```
SELECT * From fruits LIMIT 4;
```

查询结果如下：

```
+------+--------+-----------+-----------+
| f_id |  s_id  | f_name    | f_price   |
+------+--------+-----------+-----------+
| a1   |  101 | apple     |    5.20   |
| a2   |  103 | apricot   |    2.20   |
| b1   |  101 | blackberry |   10.20   |
| b2   |  104 | berry     |    7.60   |
+------+-------+-----------+-----------+
```

由结果可以看到，该语句没有指定返回记录的“位置偏移量”参数，显示结果从第一行开始，“行数”参数为 4，因此返回的结果为表中的前 4 行记录。

如果指定返回记录的开始位置，则返回结果为从“位置偏移量”参数开始的指定行数，“行数”参数指定返回的记录条数。

【例 7.33】在 fruits 表中，使用 LIMIT 子句，返回从第 5 个记录开始的，行数长度为 3 的记录，SQL 语句如下：

```
SELECT * From fruits LIMIT 4, 3;
```

查询结果如下：

```
mysql> SELECT * From fruits LIMIT 4, 3;
+------+---------+---------+-----------+
| f_id |  s_id  | f_name | f_price       |
+------+---------+---------+-----------+
| b5   |  107  | xxxx          |    3.60          |
| bs1  |  102  | orange        |   11.20 |
| bs2  |  105  | melon         |   8.20  |
+------+---------+---------+-----------+
```

由结果可以看到，该语句指示 MySQL 返回从第 5 条记录行开始之后的 3 条记录。第一个数字‘4’表示从第 5 行开始（位置偏移量从 0 开始，第 5 行的位置偏移量为 4），第二个数字 3 表示返回的行数。

所以，带一个参数的 LIMIT 指定从查询结果的首行开始，唯一的参数表示返回的行数，即“LIMIT n”与“LIMIT 0,n”等价。带两个参数的 LIMIT 可以返回从任何一个位置开始的指定的行数。

返回第一行时，位置偏移量是 0。因此，“LIMIT 1, 1”将返回第二行，而不是第一行。

MySQL 5.7 中可以使用“LIMIT 4 OFFSET 3”，意思是获取从第 5 条记录开始后面的 3 条记录，和“LIMIT 4,3;”返回的结果相同。

7.3 使用聚合函数查询

有时候并不需要返回实际表中的数据，而只是对数据进行总结。MySQL 提供一些查询功能，可以对获取的数据进行分析和报告。这些函数的功能有：计算数据表中记录行数的总数、计算某个字段列下数据的总和，以及计算表中某个字段下的最大值、最小值或者平均值。本节将介绍这些函数以及如何使用它们。这些聚合函数的名称和作用如表 7.2 所示。

表 7.2 MySQL 聚合函数

函数	作用
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值的和

接下来，将详细介绍各个函数的使用方法。

7.3.1 COUNT()函数

COUNT()函数统计数据表中包含的记录行的总数，或者根据查询结果返回列中包含的数据行数。其使用方法有两种：

- COUNT(*) 计算表中总的行数，不管某列有数值或者为空值。
- COUNT(字段名)计算指定列下总的行数，计算时将忽略空值的行。

【例 7.34】查询 customers 表中总的行数，SQL 语句如下：

```
mysql> SELECT COUNT(*) AS cust_num
    -> FROM customers;
+----------+
| cust_num |
+----------+
|        4 |
+----------+
```

由查询结果可以看到，COUNT(*)返回 customers 表中记录的总行数，不管其值是什么。返回的总数的名称为 cust_num。

【例 7.35】查询 customers 表中有电子邮箱的顾客的总数，SQL 语句如下：

```
mysql> SELECT COUNT(c_email) AS email_num
    -> FROM customers;
+-------------+
| email_num |
+-------------+
|    3    |
+-------------+
```

由查询结果可以看到，表中 5 个 customer 只有 3 个有 email，customer 的 email 为空值 NULL 的记录没有被 COUNT()函数计算。

两个例子中不同的数值，说明了两种方式在计算总数的时候对待 NULL 值的方式不同。即指定列的值为空的行被 COUNT()函数忽略，但是如果不指定列，而在 COUNT()函数中使用星号“*”，则所有记录都不忽略。

前面介绍分组查询的时候，介绍了 COUNT()函数与 GROUP BY 关键字一起使用，用来计算不同分组中的记录总数。

【例 7.36】在 orderitems 表中，使用 COUNT()函数统计不同订单号中订购的水果种类，SQL 语句如下：

```
mysql> SELECT o_num, COUNT(f_id)
    -> FROM orderitems
    -> GROUP BY o_num;
+-------+----------------+
| o_num | COUNT(f_id) |
+-------+----------------+
| 30001 |       4    |
| 30002 |       1    |
| 30003 |       1    |
| 30004 |       1    |
| 30005 |       4    |
+-------+----------------+
```

查询结果可以看到，GROUP BY 关键字先按照订单号进行分组，然后计算每个分组中的总记录数。

7.3.2 SUM()函数

SUM()是一个求总和的函数，返回指定列值的总和。

【例 7.37】在 orderitems 表中查询 30005 号订单一共购买的水果总量，SQL 语句如下：

```
mysql> SELECT SUM(quantity) AS items_total
```

```
    -> FROM orderitems
    -> WHERE o_num = 30005;
+--------------+
| items_total |
+--------------+
|     30       |
+--------------+
```

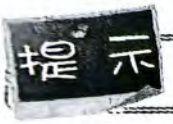
由查询结果可以看到，SUM(quantity)函数返回订单中所有水果数量之和，WHERE 子句指定查询的订单号为 30005。

SUM()可以与 GROUP BY 一起使用，来计算每个分组的总和。

【例 7.38】在 orderitems 表中，使用 SUM()函数统计不同订单号中订购的水果总量，SQL 语句如下：

```
mysql> SELECT o_num, SUM(quantity) AS items_total
    -> FROM orderitems
    -> GROUP BY o_num;
+-------+---------------+
| o_num | items_total   |
+-------+---------------+
| 30001 |     33        |
| 30002 |      2        |
| 30003 |    100        |
| 30004 |     50        |
| 30005 |     30        |
+-------+---------------+
```

由查询结果可以看到，GROUP BY 按照订单号 o_num 进行分组，SUM()函数计算每个分组中订购的水果的总量。



SUM()函数在计算时，忽略列值为 NULL 的行。

7.3.3 AVG()函数

AVG()函数通过计算返回的行数和每一行数据的和，求得指定列数据的平均值。

【例 7.39】在 fruits 表中，查询 s_id=103 的供应商的水果价格的平均值，SQL 语句如下：

```
mysql> SELECT AVG(f_price) AS avg_price
    -> FROM fruits
    -> WHERE s_id = 103;
+-----------+
| avg_price |
+-----------+
|  5.700000 |
```

```
+------------+
```

该例中，查询语句增加了一个 WHERE 子句，并且添加了查询过滤条件，只查询 s_id = 103 的记录中的 f_price。因此，通过 AVG()函数计算的结果只是指定的供应商水果的价格平均值，而不是市场上所有水果的价格的平均值。

AVG()可以与 GROUP BY 一起使用，来计算每个分组的平均值。

【例 7.40】在 fruits 表中，查询每一个供应商的水果价格的平均值，SQL 语句如下：

```
mysql> SELECT s_id,AVG(f_price) AS avg_price
    -> FROM fruits
    -> GROUP BY s_id;
+------+--------------+
| s_id |  avg_price |
+------+--------------+
|  101 |  6.200000 |
|  102 |  8.933333 |
|  103 |  5.700000 |
|  104 |  7.000000 |
|  105 |  7.466667 |
|  106 | 15.700000 |
|  107 |  3.600000 |
+------+--------------+
```

GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组的平均值，这种分组求平均值的方法非常有用，例如：求不同班级学生成绩的平均值，求不同部门工人的平均工资，求各地的年平均气温等等。

AVG()函数使用时，其参数为要计算的列名称，如果要得到多个列的多个平均值，则需要在每一列上使用 AVG()函数。

7.3.4 MAX()函数

MAX()返回指定列中的最大值。

【例 7.41】在 fruits 表中查找市场上价格最高的水果值，SQL 语句如下：

```
mysql>SELECT MAX(f_price) AS max_price FROM fruits;
+-------------+
| max_price |
+-------------+
|  15.70  |
+-------------+
```

由结果可以看到，MAX()函数查询出了 f_price 字段的最大值 15.70。

MAX()也可以和 GROUP BY 关键字一起使用，求每个分组中的最大值。

【例 7.42】在 fruits 表中查找不同供应商提供的价格最高的水果值，SQL 语句如下：

```
mysql> SELECT s_id, MAX(f_price) AS max_price
    -> FROM fruits
-> GROUP BY s_id;
+------+-----------+
| s_id | max_price |
+------+-----------+
|  101 |   10.20 |
|  102 |   11.20 |
|  103 |    9.20 |
|  104 |    7.60 |
|  105 |   11.60 |
|  106 |   15.70 |
|  107 |    3.60 |
+------+-----------+
```

由结果可以看到，GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组中的最大值。

MAX()函数不仅适用于查找数值类型，也可应用于字符类型。

【例 7.43】在 fruits 表中查找 f_name 的最大值，SQL 语句如下：

```
mysql> SELECT MAX(f_name) FROM fruits;
+-----------------+
| MAX(f_name) |
+-----------------+
| xxxx        |
+-----------------+
```

由结果可以看到，MAX()函数可以对字母进行大小判断，并返回最大的字符或者字符串值。

MAX()函数除了用来找出最大的列值或日期值之外，还可以返回任意列中的最大值，包括返回字符类型的最大值。在对字符类型数据进行比较时，按照字符的 ASCII 码值大小进行比较，从 a~z，a 的 ASCII 码最小，z 的最大。在比较时，先比较第一个字母，如果相等，继续比较下一个字符，一直到两个字符不相等或者字符结束为止。例如，‘b’与‘t’比较时，‘t’为最大值；“bcd”与“bca”比较时，“bcd”为最大值。

7.3.5 MIN()函数

MIN()返回查询列中的最小值。

【例 7.44】在 fruits 表中查找市场上价格最低的水果值，SQL 语句如下：

```
mysql>SELECT MIN(f_price) AS min_price FROM fruits;
+-----------+
```

```
| min_price |
+-----------+
|   2.20    |
+-----------+
```

由结果可以看到，MIN ()函数查询出了 f_price 字段的最小值 2.20。

MIN()也可以和 GROUP BY 关键字一起使用，求出每个分组中的最小值。

【例 7.45】在 fruits 表中查找不同供应商提供的价格最低的水果值，SQL 语句如下：

```
mysql>  SELECT s_id, MIN(f_price) AS min_price
    -> FROM fruits
    -> GROUP BY s_id;
+------+-----------+
| s_id | min_price |
+------+-----------+
|  101 |    3.20 |
|  102 |    5.30 |
|  103 |    2.20 |
|  104 |    6.40 |
|  105 |    2.60 |
|  106 |   15.70 |
|  107 |    3.60 |
+------+-----------+
```

由结果可以看到，GROUP BY 关键字根据 s_id 字段对记录进行分组，然后计算出每个分组中的最小值。

MIN()函数与 MAX()函数类似，不仅适用于查找数值类型，也可应用于字符类型。

7.4 连接查询

连接是关系数据库模型的主要特点。连接查询是关系数据库中最主要的查询，主要包括内连接、外连接等。通过连接运算符可以实现多个表查询。在关系数据库管理系统中，表建立时各数据之间的关系不必确定，常把一个实体的所有信息存放在一个表中。当查询数据时，通过连接操作查询出存放在多个表中的不同实体的信息。当两个或多个表中存在相同意义的字段时，便可以通过这些字段对不同的表进行连接查询。本节将介绍多表之间的内连接查询、外连接查询以及复合条件连接查询。

7.4.1 内连接查询

内连接（INNER JOIN）使用比较运算符进行表间某（些）列数据的比较操作，并列出这些表中与连接条件相匹配的数据行，组合成新记录，也就是说，在内连接查询中，只有满足条

件的记录才能出现在结果关系中。

为了演示的需要，首先创建数据表 suppliers，SQL 语句如下：

```
CREATE TABLE suppliers
(
  s_id      int      NOT NULL AUTO_INCREMENT,
  s_name    char(50) NOT NULL,
  s_city    char(50) NULL,
  s_zip     char(10) NULL,
  s_call    CHAR(50) NOT NULL,
  PRIMARY KEY (s_id)
) ;
```

插入需要演示的数据，SQL 语句如下：

```
INSERT INTO suppliers(s_id, s_name,s_city,  s_zip, s_call)
VALUES(101,'FastFruit Inc.','Tianjin','300000','48075'),
(102,'LT Supplies','Chongqing','400000','44333'),
(103,'ACME','Shanghai','200000','90046'),
(104,'FNK Inc.','Zhongshan','528437','11111'),
(105,'Good Set','Taiyuang','030000', '22222'),
(106,'Just Eat Ours','Beijing','010', '45678'),
(107,'DK Inc.','Zhengzhou','450000', '33332');
```

【例 7.46】在 fruits 表和 suppliers 表之间使用内连接查询。

查询之前，查看两个表的结构：

```
mysql> DESC fruits;
+---------+-------------+---------+-------+-----------+---------+
| Field   | Type        | Null    | Key | Default | Extra   |
+---------+-------------+---------+-------+-----------+---------+
| f_id    | char(10)    | NO      | PRI | NULL    |         |
| s_id    | int(11)     | NO      |     | NULL    |         |
| f_name  | char(255)   | NO      |     | NULL    |         |
| f_price | decimal(8,2)| NO      |     | NULL    |         |
+---------+-------------+---------+-------+-----------+---------+

mysql> DESC suppliers;
+--------+----------+------+-----+---------+----------------+
| Field  | Type     | Null | Key | Default | Extra          |
+--------+----------+------+-----+---------+----------------+
| s_id   | int(11)  | NO   | PRI | NULL    | auto_increment |
| s_name | char(50) | NO   |     | NULL    |                |
| s_city | char(50) | YES  |     | NULL    |                |
| s_zip  | char(10) | YES  |     | NULL    |                |
| s_call | char(50) | NO   |     | NULL    |                |
```

```
+--------+----------+--------+---------+--------------+-------------------
-------+
```

由结果可以看到，fruits 表和 suppliers 表中都有相同数据类型的字段 s_id，两个表通过 s_id 字段建立联系。接下来从 fruits 表中查询 f_name、f_price 字段，从 suppliers 表中查询 s_id、s_name，SQL 语句如下：

```
mysql> SELECT suppliers.s_id, s_name,f_name, f_price
    -> FROM fruits ,suppliers
    -> WHERE fruits.s_id = suppliers.s_id;
+------+---------------+---------------+----------+
| s_id | s_name        | f_name        | f_price |
+------+---------------+---------------+----------+
|  101 | FastFruit Inc. | apple        |    5.20 |
|  103 | ACME          | apricot       |    2.20 |
|  101 | FastFruit Inc. | blackberry   |   10.20 |
|  104 | FNK Inc.      | berry         |    7.60 |
|  107 | DK Inc.       | xxxx          |    3.60 |
|  102 | LT Supplies   | orange        |   11.20 |
|  105 | Good Set      | melon         |    8.20 |
|  101 | FastFruit Inc. | cherry       |    3.20 |
|  104 | FNK Inc.      | lemon         |    6.40 |
|  106 | Just Eat Ours | mango         |   15.70 |
|  105 | Good Set      | xbabay        |    2.60 |
|  105 | Good Set      | xxtt          |   11.60 |
|  103 | ACME          | coconut       |    9.20 |
|  102 | LT Supplies   | banana        |   10.30 |
|  102 | LT Supplies   | grape         |    5.30 |
|  107 | DK Inc.       | xbababa       |    3.60 |
+------+---------------+---------------+----------+
```

在这里，SELECT 语句与前面所介绍的一个最大的差别是：SELECT 后面指定的列分别属于两个不同的表，（f_name，f_price）在表 fruits 中，而另外两个字段在表 supplies 中；同时 FROM 子句列出了两个表 fruits 和 suppliers。WHERE 子句在这里作为过滤条件，指明只有两个表中的 s_id 字段值相等的时候才符合连接查询的条件。从返回的结果可以看到，显示的记录是由两个表中不同列值组成的新记录。

因为 fruits 表和 suppliers 表中有相同的字段 s_id，因此在比较的时候，需要完全限定表名（格式为“表名.列名”），如果只给出 s_id，MySQL 将不知道指的是哪一个，并返回错误信息。

下面的内连接查询语句返回与前面完全相同的结果。

【例 7.47】在 fruits 表和 suppliers 表之间，使用 INNER JOIN 语法进行内连接查询，SQL 语句如下：

```
mysql> SELECT suppliers.s_id, s_name,f_name, f_price
    -> FROM fruits INNER JOIN suppliers
    -> ON fruits.s_id = suppliers.s_id;
+------+-----------------+-----------+-----------+
| s_id | s_name          | f_name    | f_price   |
+------+-----------------+-----------+-----------+
|  101 | FastFruit Inc.  | apple     |    5.20   |
|  103 | ACME            | apricot   |    2.20   |
|  101 | FastFruit Inc.  | blackberry|   10.20   |
|  104 | FNK Inc.        | berry     |    7.60   |
|  107 | DK Inc.         | xxxx      |    3.60   |
|  102 | LT Supplies     | orange    |   11.20   |
|  105 | Good Set        | melon     |    8.20   |
|  101 | FastFruit Inc.  | cherry    |    3.20   |
|  104 | FNK Inc.        | lemon     |    6.40   |
|  106 | Just Eat Ours   | mango     |   15.70   |
|  105 | Good Set        | xbabay    |    2.60   |
|  105 | Good Set        | xxtt      |   11.60   |
|  103 | ACME            | coconut   |    9.20   |
|  102 | LT Supplies     | banana    |   10.30   |
|  102 | LT Supplies     | grape     |    5.30   |
|  107 | DK Inc.         | xbababa   |    3.60   |
+------+-----------------+-----------+-----------+
```

在这里的查询语句中，两个表之间的关系通过 INNER JOIN 指定。使用这种语法的时候，连接的条件使用 ON 子句给出而不是 WHERE，ON 和 WHERE 后面指定的条件相同。

使用 WHERE 子句定义连接条件比较简单明了，而 INNER JOIN 语法是 ANSI SQL 的标准规范，使用 INNER JOIN 连接语法能够确保不会忘记连接条件，而且，WHERE 子句在某些时候会影响查询的性能。

如果在一个连接查询中，涉及的两个表都是同一个表，这种查询称为自连接查询。自连接是一种特殊的内连接，它是指相互连接的表在物理上为同一张表，但可以在逻辑上分为两张表。

【例 7.48】查询供应 f_id= 'a1'的水果供应商提供的水果种类，SQL 语句如下：

```
mysql> SELECT f1.f_id, f1.f_name
    -> FROM fruits AS f1, fruits AS f2
    -> WHERE f1.s_id = f2.s_id AND f2.f_id = 'a1';
+------+-----------+
| f_id | f_name    |
+------+-----------+
| a1   | apple     |
```

```
| b1   | blackberry |
| c0   | cherry     |
+------+------------+
```

此处查询的两个表是相同的表，为了防止产生二义性，对表使用了别名，ftuits 表第 1 次出现的别名为 f1，第 2 次出现的别名为 f2，使用 SELECT 语句返回列时明确指出返回以 f1 为前缀的列的全名，WHERE 连接两个表，并按照第 2 个表的 f_id 对数据进行过滤，返回所需数据。

7.4.2 外连接查询

外连接查询将查询多个表中相关联的行，内连接时，返回查询结果集合中的仅是符合查询条件和连接条件的行。但有时候需要包含没有关联的行中数据，即返回查询结果集合中的不仅包含符合连接条件的行，而且还包括左表（左外连接或左连接）、右表（右外连接或右连接）或两个连接表（全外连接）中的所有数据行。外连接分为左外连接或左连接和右外连接或右连接：

- LEFT JOIN（左连接）：返回包括左表中的所有记录和右表中连接字段相等的记录。
- RIGHT JOIN（右连接）：返回包括右表中的所有记录和左表中连接字段相等的记录。

1. LEFT JOIN（左连接）

左连接的结果包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是连接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果行中，右表的所有选择列表列均为空值。

首先创建表 orders，SQL 语句如下：

```
CREATE TABLE orders
(
  o_num  int      NOT NULL AUTO_INCREMENT,
  o_date datetime NOT NULL,
  c_id   int      NOT NULL,
  PRIMARY KEY (o_num)
) ;
```

插入需要演示的数据，SQL 语句如下：

```
INSERT INTO orders(o_num, o_date, c_id)
VALUES(30001, '2008-09-01', 10001),
(30002, '2008-09-12', 10003),
(30003, '2008-09-30', 10004),
(30004, '2008-10-03', 10005),
(30005, '2008-10-08', 10001);
```

【例 7.49】在 customers 表和 orders 表中，查询所有客户，包括没有订单的客户，SQL 语

句如下：

```
mysql> SELECT customers.c_id, orders.o_num
    -> FROM customers LEFT OUTER JOIN orders
    -> ON customers.c_id = orders.c_id;
+-------+-------+
| c_id  | o_num |
+-------+-------+
| 10001 | 30001 |
| 10001 | 30005 |
| 10002 | NULL  |
| 10003 | 30002 |
| 10004 | 30003 |
+-------+-------+
```

结果显示了 5 条记录，ID 等于 10002 的客户目前并没有下订单，所以对应的 orders 表中并没有该客户的订单信息，所以该条记录只取出了 customers 表中相应的值，而从 orders 表中取出的值为空值 NULL。

2. RIGHT JOIN（右连接）

右连接是左连接的反向连接，将返回右表的所有行。如果右表的某行在左表中没有匹配行，左表将返回空值。

【例 7.50】在 customers 表和 orders 表中，查询所有订单，包括没有客户的订单，SQL 语句如下：

```
mysql> SELECT customers.c_id, orders.o_num
    -> FROM customers RIGHT OUTER JOIN orders
    -> ON customers.c_id = orders.c_id;
+-------+-------+
| c_id  | o_num |
+-------+-------+
| 10001 | 30001 |
| 10003 | 30002 |
| 10004 | 30003 |
| NULL  | 30004 |
| 10001 | 30005 |
+-------+-------+
```

结果显示了 5 条记录，订单号等于 30004 的订单的客户可能由于某种原因取消了该订单，对应的 customers 表中并没有该客户的信息，所以该条记录只取出了 ordes 表中相应的值，而从 customers 表中取出的值为空值 NULL。

7.4.3 复合条件连接查询

复合条件连接查询是在连接查询的过程中，通过添加过滤条件，限制查询的结果，使查询的结果更加准确。

【例 7.51】在 customers 表和 orders 表中，使用 INNER JOIN 语法查询 customers 表中 ID 为 10001 的客户的订单信息，SQL 语句如下：

```
mysql> SELECT customers.c_id, orders.o_num
    -> FROM customers INNER JOIN orders
    -> ON customers.c_id = orders.c_id AND customers.c_id = 10001;
+-------+-------+
| c_id  | o_num |
+-------+-------+
| 10001 | 30001 |
| 10001 | 30005 |
+-------+-------+
```

结果显示，在连接查询时指定查询客户 ID 为 10001 的订单信息，添加了过滤条件之后返回的结果将会变少，因此返回结果只有两条记录。

使用连接查询，并对查询的结果进行排序。

【例 7.52】在 fruits 表和 suppliers 表之间，使用 INNER JOIN 语法进行内连接查询，并对查询结果排序，SQL 语句如下：

```
mysql> SELECT suppliers.s_id, s_name,f_name, f_price
    -> FROM fruits INNER JOIN suppliers
    -> ON fruits.s_id = suppliers.s_id
    -> ORDER BY fruits.s_id;
+------+---------------+-------------+----------+
| s_id | s_name        | f_name      | f_price  |
+------+---------------+-------------+----------+
|  101 | FastFruit Inc. | apple      |    5.20 |
|  101 | FastFruit Inc. | blackberry |   10.20 |
|  101 | FastFruit Inc. | cherry     |    3.20 |
|  102 | LT Supplies    | grape      |    5.30 |
|  102 | LT Supplies    | banana     |   10.30 |
|  102 | LT Supplies    | orange     |   11.20 |
|  103 | ACME           | apricot    |    2.20 |
|  103 | ACME           | coconut    |    9.20 |
|  104 | FNK Inc.       | lemon      |    6.40 |
|  104 | FNK Inc.       | berry      |    7.60 |
|  105 | Good Set       | xbabay     |    2.60 |
|  105 | Good Set       | xxtt       |   11.60 |
|  105 | Good Set       | melon      |    8.20 |
|  106 | Just Eat Ours  | mango      |   15.70 |
```

```
| 107 | DK Inc.          | xxxx         |  3.60 |
| 107 | DK Inc.          | xbababa      |  3.60 |
+-----+------------------+--------------+-------+
```

由结果可以看到，内连接查询的结果按照 suppliers.s_id 字段进行了升序排序。

7.5 子查询

子查询指一个查询语句嵌套在另一个查询语句内部的查询，这个特性从 MySQL 4.1 开始引入。在 SELECT 子句中先计算子查询，子查询结果作为外层另一个查询的过滤条件，查询可以基于一个表或者多个表。子查询中常用的操作符有 ANY（SOME）、ALL、IN、EXISTS。子查询可以添加到 SELECT、UPDATE 和 DELETE 语句中，而且可以进行多层嵌套。子查询中也可以使用比较运算符，如“<”、“<=”、“>”、“>=”和“!=”等。本节将介绍如何在 SELECT 语句中嵌套子查询。

7.5.1 带 ANY、SOME 关键字的子查询

ANY 和 SOME 关键字是同义词，表示满足其中任一条件，它们允许创建一个表达式对子查询的返回值列表进行比较，只要满足内层子查询中的任何一个比较条件，就返回一个结果作为外层查询的条件。

下面定义两个表 tbl1 和 tbl2：

```
CREATE table tbl1 ( num1 INT NOT NULL);
CREATE table tbl2 ( num2 INT NOT NULL);
```

分别向两个表中插入数据：

```
INSERT INTO tbl1 values(1), (5), (13), (27);
INSERT INTO tbl2 values(6), (14), (11), (20);
```

ANY 关键字接在一个比较操作符的后面，表示若与子查询返回的任何值比较为 TRUE，则返回 TRUE。

【例 7.53】返回 tbl2 表的所有 num2 列，然后将 tbl1 中的 num1 的值与之进行比较，只要大于 num2 的任何 1 个值，即为符合查询条件的结果。

```
mysql> SELECT num1 FROM tbl1 WHERE num1 > ANY (SELECT num2 FROM tbl2);
+------+
| num1 |
+------+
|   13 |
|   27 |
```

```
+-------+
```

在子查询中，返回的是 tbl2 表的所有 num2 列结果（6,14,11,20），然后将 tbl1 中的 num1 列的值与之进行比较，只要大于 num2 列的任意一个数即为符合条件的结果。

7.5.2 带 ALL 关键字的子查询

ALL 关键字与 ANY 和 SOME 不同，使用 ALL 时需要同时满足所有内层查询的条件。例如，修改前面的例子，用 ALL 关键字替换 ANY。

ALL 关键字接在一个比较操作符的后面，表示与子查询返回的所有值比较为 TRUE，则返回 TRUE。

【例 7.54】返回 tbl1 表中比 tbl2 表 num2 列所有值都大的值，SQL 语句如下：

```
mysql> SELECT num1 FROM tbl1 WHERE num1 > ALL (SELECT num2 FROM tbl2);
+-------+
| num1 |
+-------+
|   27 |
+-------+
```

在子查询中，返回的是 tbl2 的所有 num2 列结果（6,14,11,20），然后将 tbl1 中的 num1 列的值与之进行比较，大于所有 num2 列值的 num1 值只有 27，因此返回结果为 27。

7.5.3 带 EXISTS 关键字的子查询

EXISTS 关键字后面的参数是一个任意的子查询，系统对子查询进行运算以判断它是否返回行，如果至少返回一行，那么 EXISTS 的结果为 true，此时外层查询语句将进行查询；如果子查询没有返回任何行，那么 EXISTS 返回的结果是 false，此时外层语句将不进行查询。

【例 7.55】查询 suppliers 表中是否存在 s_id=107 的供应商，如果存在，则查询 fruits 表中的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits
    -> WHERE EXISTS
    -> (SELECT s_name FROM suppliers WHERE s_id = 107);
+------+------+--------------+---------+
| f_id | s_id | f_name       | f_price |
+------+------+--------------+---------+
| a1   |  101 | apple        |    5.20 |
| a2   |  103 | apricot      |    2.20 |
| b1   |  101 | blackberry   |   10.20 |
| b2   |  104 | berry        |    7.60 |
| b5   |  107 | xxxx         |    3.60 |
| bs1  |  102 | orange       |   11.20 |
```

```
| bs2  |  105 | melon      |   8.20 |
| c0   |  101 | cherry     |   3.20 |
| l2   |  104 | lemon      |   6.40 |
| m1   |  106 | mango      |  15.70 |
| m2   |  105 | xbabay     |   2.60 |
| m3   |  105 | xxtt       |  11.60 |
| o2   |  103 | coconut    |   9.20 |
| t1   |  102 | banana     |  10.30 |
| t2   |  102 | grape      |   5.30 |
| t4   |  107 | xbababa    |   3.60 |
+------+------+------------+--------+
```

由结果可以看到，内层查询结果表明 suppliers 表中存在 s_id=107 的记录，因此 EXISTS 表达式返回 true；外层查询语句接收 true 之后对表 fruits 进行查询，返回所有的记录。

EXISTS 关键字可以和条件表达式一起使用。

【例 7.56】查询 suppliers 表中是否存在 s_id=107 的供应商，如果存在，则查询 fruits 表中的 f_price 大于 10.20 的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits
    -> WHERE f_price>10.20 AND EXISTS
    -> (SELECT s_name FROM suppliers WHERE s_id = 107);
+------+------+--------+---------+
| f_id | s_id | f_name | f_price |
+--------+-------+--------+---------+
| bs1  |  102 | orange |  11.20 |
| m1   |  106 | mango  |  15.70 |
| m3   |  105 | xxtt   |  11.60 |
| t1   |  102 | banana |  10.30 |
+--------+-------+--------+---------+
```

由结果可以看到，内层查询结果表明 suppliers 表中存在 s_id=107 的记录，因此 EXISTS 表达式返回 true；外层查询语句接收 true 之后根据查询条件 f_price > 10.20 对 fruits 表进行查询，返回结果为 4 条 f_price 大于 10.20 的记录。

NOT EXISTS 与 EXISTS 使用方法相同，返回的结果相反。子查询如果至少返回一行，那么 NOT EXISTS 的结果为 false，此时外层查询语句将不进行查询；如果子查询没有返回任何行，那么 NOT EXISTS 返回的结果是 true，此时外层语句将进行查询。

【例 7.57】查询 suppliers 表中是否存在 s_id=107 的供应商，如果不存在则查询 fruits 表中的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits
    -> WHERE NOT EXISTS
    -> (SELECT s_name FROM suppliers WHERE s_id = 107);
Empty set (0.00 sec)
```

查询语句 SELECT s_name FROM suppliers WHERE s_id = 107，对 suppliers 表进行查询返回了一条记录，NOT EXISTS 表达式返回 false，外层表达式接收 false，将不再查询 fruits 表中的记录。

EXISTS 和 NOT EXISTS 的结果只取决于是否会返回行，而不取决于这些行的内容，所以这个子查询输入列表通常是无关紧要的。

7.5.4 带 IN 关键字的子查询

IN 关键字进行子查询时，内层查询语句仅仅返回一个数据列，这个数据列里的值将提供给外层查询语句进行比较操作。

【例 7.58】在 orderitems 表中查询 f_id 为 c0 的订单号，并根据订单号查询具有订单号的客户 c_id，SQL 语句如下：

```
mysql> SELECT c_id FROM orders WHERE o_num IN
    -> (SELECT o_num  FROM orderitems WHERE f_id = 'c0');
+--------+
| c_id   |
+--------+
| 10004  |
| 10001  |
+--------+
```

查询结果的 c_id 有两个值，分别为 10001 和 10004。上述查询过程可以分步执行，首先内层子查询查出 orderitems 表中符合条件的订单号，单独执行内查询，查询结果如下：

```
mysql> SELECT o_num  FROM orderitems WHERE f_id = 'c0';
+--------+
| o_num  |
+--------+
| 30003  |
| 30005  |
+--------+
```

可以看到，符合条件的 o_num 列的值有两个：30003 和 30005，然后执行外层查询，在 orders 表中查询订单号等于 30003 或 30005 的客户 c_id。嵌套子查询语句还可以写为如下形式，实现相同的效果：

```
mysql> SELECT c_id FROM orders WHERE o_num IN (30003, 30005);
+--------+
| c_id   |
+--------+
| 10004  |
| 10001  |
```

```
+--------+
```

这个例子说明在处理 SELECT 语句的时候，MySQL 实际上执行了两个操作过程，即先执行内层子查询，再执行外层查询，内层子查询的结果作为外部查询的比较条件。

SELECT 语句中可以使用 NOT IN 关键字，其作用与 IN 正好相反。

【例 7.59】与前一个例子类似，但是在 SELECT 语句中使用 NOT IN 关键字，SQL 语句如下：

```
mysql> SELECT c_id FROM orders WHERE o_num NOT IN
    -> (SELECT o_num  FROM orderitems WHERE f_id = 'c0');
+--------+
| c_id   |
+--------+
| 10001  |
| 10003  |
| 10005  |
+--------+
```

这里返回的结果有 3 条记录，由前面可以看到，子查询返回的订单值有两个，即 30003 和 30005，但为什么这里还有值为 10001 的 c_id 呢？这是因为 c_id 等于 10001 的客户的订单不止一个，可以查看订单表 orders 中的记录。

```
mysql> SELECT * FROM orders;
+-------+---------------------+-------+
| o_num | o_date              | c_id  |
+-------+---------------------+-------+
| 30001 | 2008-09-01 00:00:00 | 10001 |
| 30002 | 2008-09-12 00:00:00 | 10003 |
| 30003 | 2008-09-30 00:00:00 | 10004 |
| 30004 | 2008-10-03 00:00:00 | 10005 |
| 30005 | 2008-10-08 00:00:00 | 10001 |
+-------+---------------------+-------+
```

可以看到，虽然排除了订单号为 30003 和 30005 的客户 c_id，但是 o_num 为 30001 的订单与 30005 都是 10001 号客户的订单。所以结果中只是排除了订单号，但是仍然有可能选择同一个客户。

子查询的功能也可以通过连接查询完成，但是子查询使得MySQL代码更容易阅读和编写。

7.5.5 带比较运算符的子查询

在前面介绍的带 ANY、ALL 关键字的子查询时使用了“>”比较运算符，子查询时还可以使用其他的比较运算符，如“<”、“<=”、“=”、“>=”和“!=”等。

【例 7.60】在 suppliers 表中查询 s_city 等于“Tianjin”的供应商 s_id，然后在 fruits 表中查询所有该供应商提供的水果的种类，SQL 语句如下：

```
SELECT s_id, f_name FROM fruits
WHERE s_id =
(SELECT s1.s_id FROM suppliers AS s1 WHERE s1.s_city = 'Tianjin');
```

该嵌套查询首先在 suppliers 表中查找 s_city 等于 Tianjin 的供应商的 s_id，单独执行子查询查看 s_id 的值，执行下面的操作过程：

```
mysql> SELECT s1.s_id FROM suppliers AS s1 WHERE s1.s_city = 'Tianjin';
+------+
| s_id |
+------+
|  101 |
+------+
```

然后在外层查询时，在 fruits 表中查找 s_id 等于 101 的供应商提供的水果的种类，查询结果如下：

```
mysql> SELECT s_id, f_name FROM fruits
    -> WHERE s_id =
    -> (SELECT s1.s_id FROM suppliers AS s1 WHERE s1.s_city = 'Tianjin');
+------+------------+
| s_id | f_name     |
+------+------------+
|  101 | apple      |
|  101 | blackberry |
|  101 | cherry     |
+------+------------+
```

结果表明，“Tianjin”地区的供应商提供的水果种类有 3 种，分别为“apple”、“blackberry”、“cherry”。

【例 7.61】在 suppliers 表中查询 s_city 等于“Tianjin”的供应商 s_id，然后在 fruits 表中查询所有非该供应商提供的水果的种类，SQL 语句如下：

```
mysql> SELECT s_id, f_name FROM fruits
    -> WHERE s_id <>
    -> (SELECT s1.s_id FROM suppliers AS s1 WHERE s1.s_city = 'Tianjin');
+------+---------+
| s_id | f_name  |
+------+---------+
|  103 | apricot |
|  104 | berry   |
|  107 | xxxx    |
|  102 | orange  |
|  105 | melon   |
|  104 | lemon   |
```

```
|  106 | mango   |
|  105 | xbabay  |
|  105 | xxtt    |
|  103 | coconut |
|  102 | banana  |
|  102 | grape   |
|  107 | xbababa |
+------+---------+
```

该嵌套查询执行过程与前面相同，在这里使用了不等于“<>”运算符，因此返回的结果和前面正好相反。

7.6 合并查询结果

利用 UNION 关键字，可以给出多条 SELECT 语句，并将它们的结果组合成单个结果集。合并时，两个表对应的列数和数据类型必须相同。各个 SELECT 语句之间使用 UNION 或 UNION ALL 关键字分隔。UNION 不使用关键字 ALL，执行的时候删除重复的记录，所有返回的行都是唯一的；使用关键字 ALL 的作用是不删除重复行也不对结果进行自动排序。基本语法格式如下：

```
SELECT column,... FROM table1
UNION [ALL]
SELECT column,... FROM table2
```

【例 7.62】查询所有价格小于 9 的水果的信息，查询 s_id 等于 101 和 103 所有的水果的信息，使用 UNION 连接查询结果，SQL 语句如下：

```
SELECT s_id, f_name, f_price
FROM fruits
WHERE f_price < 9.0
UNION ALL
SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id IN(101,103);
```

合并查询结果如下：

```
+------+--------------+---------+
| s_id | f_name       | f_price |
+------+--------------+---------+
|  101 | apple        |    5.20 |
|  103 | apricot      |    2.20 |
|  104 | berry        |    7.60 |
|  107 | xxxx         |    3.60 |
|  105 | melon        |    8.20 |
|  101 | cherry       |    3.20 |
|  104 | lemon        |    6.40 |
|  105 | xbabay       |    2.60 |
```

```
| 102 | grape      |   5.30 |
| 107 | xbababa    |   3.60 |
| 101 | apple      |   5.20 |
| 103 | apricot    |   2.20 |
| 101 | blackberry |  10.20 |
| 101 | cherry     |   3.20 |
| 103 | coconut    |   9.20 |
+------+------------+--------+
```

如前所述，UNION 将多个 SELECT 语句的结果组合成一个结果集合。可以分开查看每个 SELECT 语句的结果：

```
mysql> SELECT s_id, f_name, f_price
    -> FROM fruits
    -> WHERE f_price < 9.0;
+------+----------+---------+
| s_id | f_name   | f_price |
+------+----------+---------+
| 101  | apple    |    5.20 |
| 103  | apricot  |    2.20 |
| 104  | berry    |    7.60 |
| 107  | xxxx     |    3.60 |
| 105  | melon    |    8.20 |
| 101  | cherry   |    3.20 |
| 104  | lemon    |    6.40 |
| 105  | xbabay   |    2.60 |
| 102  | grape    |    5.30 |
| 107  | xbababa  |    3.60 |
+------+----------+---------+
10 rows in set (0.00 sec)

mysql> SELECT s_id, f_name, f_price
    -> FROM fruits
    -> WHERE s_id IN(101,103);
+------+----------+---------+
| s_id | f_name   | f_price |
+------+----------+---------+
| 101  | apple    |    5.20 |
| 103  | apricot  |    2.20 |
| 101  | blackberry|   10.20 |
| 101  | cherry   |    3.20 |
| 103  | coconut  |    9.20 |
+------+----------+---------+
5 rows in set (0.00 sec)
```

由分开查询的结果可以看到，第 1 条 SELECT 语句查询价格小于 9 的水果，第 2 条 SELECT 语句查询供应商 101 和 103 提供的水果。使用 UNION 将两条 SELECT 语句分隔开，执行完毕之后把输出结果组合成单个的结果集，并删除重复的记录。

使用 UNION ALL 包含重复的行，在前面的例子中，分开查询时，两个返回结果中有相同的记录。UNION 从查询结果集中自动去除了重复的行，如果要返回所有匹配行，而不进行删除，可以使用 UNION ALL。

【例 7.63】查询所有价格小于 9 的水果的信息，查询 s_id 等于 101 和 103 的所有水果的信息，使用 UNION ALL 连接查询结果，SQL 语句如下：

```
SELECT s_id, f_name, f_price
FROM fruits
WHERE f_price < 9.0
UNION ALL
SELECT s_id, f_name, f_price
FROM fruits
WHERE s_id IN(101,103);
```

查询结果如下：

```
+------+---------------+---------+
| s_id | f_name        | f_price |
+------+---------------+---------+
|  101 | apple         |    5.20 |
|  103 | apricot       |    2.20 |
|  104 | berry         |    7.60 |
|  107 | xxxx          |    3.60 |
|  105 | melon         |    8.20 |
|  101 | cherry        |    3.20 |
|  104 | lemon         |    6.40 |
|  105 | xbabay        |    2.60 |
|  102 | grape         |    5.30 |
|  107 | xbababa       |    3.60 |
|  101 | apple         |    5.20 |
|  103 | apricot       |    2.20 |
|  101 | blackberry    |   10.20 |
|  101 | cherry        |    3.20 |
|  103 | coconut       |    9.20 |
+------+---------------+---------+
```

由结果可以看到，这里总的记录数等于两条 SELECT 语句返回的记录数之和，连接查询结果并没有去除重复的行。

UNION 和 UNION ALL 的区别：使用 UNION ALL 的功能是不删除重复行，加上 ALL 关键字语句执行时所需要的资源少，所以尽可能地使用它，因此知道有重复行但是想保留这些行，确定查询结果中不会有重复数据或者不需要去掉重复数据的时候，应当使用 UNION ALL 以提高查询效率。

7.7 为表和字段取别名

在前面介绍分组查询、聚合函数查询和嵌套子查询章节中，读者注意到有的地方使用了 AS 关键字为查询结果中的某一列指定一个特定的名字。在内连接查询时，则对相同的表 fruits 分别指定两个不同的名字，这里可以为字段或者表取一个别名，在查询时，使用别名替代其指

定的内容，本节将介绍如何为字段和表创建别名以及如何使用别名。

7.7.1 为表取别名

当表名字很长或者执行一些特殊查询时，为了方便操作或者需要多次使用相同的表时，可以为表指定别名，用这个别名替代表原来的名称。为表取别名的基本语法格式为：

```
表名 [AS] 表别名
```

“表名”为数据库中存储的数据表的名称，“表别名”为查询时指定的表的新名称，AS 关键字为可选参数。

【例 7.64】为 orders 表取别名 o，查询 30001 订单的下单日期，SQL 语句如下：

```
SELECT * FROM orders AS o
WHERE o.o_num = 30001;
```

在这里 orders AS o 代码表示为 orders 表取别名为 o，指定过滤条件时直接使用 o 代替 orders，查询结果如下：

```
+-------+---------------------+-------+
| o_num | o_date              | c_id  |
+-------+---------------------+-------+
| 30001 | 2008-09-01 00:00:00 | 10001 |
+-------+---------------------+-------+
```

【例 7.65】为 customers 和 orders 表分别取别名，并进行连接查询，SQL 语句如下：

```
mysql> SELECT c.c_id, o.o_num
    -> FROM customers AS c LEFT OUTER JOIN orders AS o
    -> ON c.c_id = o.c_id;
+-------+-------+
| c_id  | o_num |
+-------+-------+
| 10001 | 30001 |
| 10001 | 30005 |
| 10002 | NULL  |
| 10003 | 30002 |
| 10004 | 30003 |
+-------+-------+
```

由结果看到，MySQL 可以同时为多个表取别名，而且表别名可以放在不同的位置，如 WHERE 子句、SELECT 列表、ON 子句以及 ORDER BY 子句等。

在前面介绍内连接查询时指出自连接是一种特殊的内连接，在连接查询中的两个表都是同一个表，其查询语句如下：

```
mysql> SELECT f1.f_id, f1.f_name
    -> FROM fruits AS f1, fruits AS f2
    -> WHERE f1.s_id = f2.s_id AND f2.f_id = 'a1';
+------+------------+
| f_id | f_name     |
```

```
+------+------------+
| a1   | apple      |
| b1   | blackberry |
| c0   | cherry     |
+------+------------+
```

在这里，如果不使用表别名，MySQL 将不知道引用的是哪个 fruits 表实例，这是表别名的一个非常有用的地方。

在为表取别名时，要保证不能与数据库中的其他表的名称冲突。

7.7.2 为字段取别名

在本章和前面各章节的例子中可以看到，在使用 SELECT 语句显示查询结果时，MySQL 会显示每个 SELECT 后面指定的输出列，在有些情况下，显示的列的名称会很长或者名称不够直观，MySQL 可以指定列别名，替换字段或表达式。为字段取别名的基本语法格式为：

```
列名 [AS] 列别名
```

“列名”为表中字段定义的名称，“列别名”为字段新的名称，AS 关键字为可选参数。

【例 7.66】查询 fruits 表，为 f_name 取别名 fruit_name，f_price 取别名 fruit_price，为 fruits 表取别名 f1，查询表中 f_price < 8 的水果的名称，SQL 语句如下：

```
mysql> SELECT f1.f_name AS fruit_name, f1.f_price AS fruit_price
    -> FROM fruits AS f1
    -> WHERE f1.f_price < 8;
+--------------+-------------+
| fruit_name   | fruit_price |
+--------------+-------------+
| apple        |        5.20 |
| apricot      |        2.20 |
| berry        |        7.60 |
| xxxx         |        3.60 |
| cherry       |        3.20 |
| lemon        |        6.40 |
| xbabay       |        2.60 |
| grape        |        5.30 |
| xbababa      |        3.60 |
+--------------+-------------+
```

也可以为 SELECT 子句中的计算字段取别名，例如，对使用 COUNT 聚合函数或者 CONCAT 等系统函数执行的结果字段取别名。

【例 7.67】查询 suppliers 表中字段 s_name 和 s_city，使用 CONCAT 函数连接这两个字段值，并取列别名为 suppliers_title。

如果没有对连接后的值取别名，其显示列名称将会不够直观，SQL 语句如下：

```
mysql> SELECT CONCAT(TRIM(s_name) , ' (', TRIM(s_city), ')')
```

```
    -> FROM suppliers
    -> ORDER BY s_name;
+------------------------------------------------------------+
| CONCAT(TRIM(s_name) , ' (', TRIM(s_city), ')') |
+------------------------------------------------------------+
| ACME (Shanghai)                                            |
| DK Inc. (Qingdao)                                          |
| FastFruit Inc. (Tianjin)                                   |
| FNK Inc. (Zhongshan)                                       |
| Good Set (Taiyuan)                                         |
| Just Eat Ours (Beijing)                                    |
| LT Supplies (Chongqing)                                    |
+------------------------------------------------------------+
```

由结果可以看到，显示结果的列名称为 SELECT 子句后面的计算字段，实际上计算之后的列是没有名字的，这样的结果让人很不容易理解，如果为字段取一个别名，将会使结果清晰，SQL 语句如下：

```
mysql> SELECT CONCAT(TRIM(s_name) , ' (', TRIM(s_city), ')')
    -> AS suppliers_title
    -> FROM suppliers
    -> ORDER BY s_name;
+------------------------------+
| suppliers_title              |
+------------------------------+
| ACME (Shanghai)              |
| DK Inc. (Qingdao)            |
| FastFruit Inc. (Tianjin)     |
| FNK Inc. (Zhongshan)         |
| Good Set (Taiyuan)           |
| Just Eat Ours (Beijing)      |
| LT Supplies (Chongqing)      |
+------------------------------+
```

由结果可以看到，SELECT 子句计算字段值之后增加了 AS suppliers_title，它指示 MySQL 为计算字段创建一个别名 suppliers_title，显示结果为指定的列别名，这样就增强了查询结果的可读性。

表别名只在执行查询的时候使用，并不在返回结果中显示，而列别名定义之后，将返回给客户端显示，显示的结果字段为字段列的别名。

7.8 使用正则表达式查询

正则表达式通常被用来检索或替换那些符合某个模式的文本内容，根据指定的匹配模式匹配文本中符合要求的特殊字符串。例如从一个文本文件中提取电话号码，查找一篇文章中重复的单词或者替换用户输入的某些敏感词语等等，这些地方都可以使用正则表达式。正则表达式

强大而且灵活，可以应用于非常复杂的查询。

MySQL 中使用 REGEXP 关键字指定正则表达式的字符匹配模式，表 7.3 列出了 REGEXP 操作符中常用字符匹配列表。

表 7.3 正则表达式常用字符匹配列表

选项	说明	例子	匹配值示例
^	匹配文本的开始字符	'^b'匹配以字母 b 开头的字符串	book, big, banana, bike
$	匹配文本的结束字符	'st$'匹配以 st 结尾的字符串	test, resist, persist
.	匹配任何单个字符	'b.t'匹配任何 b 和 t 之间有一个字符	bit, bat, but,bite
*	匹配零个或多个在它前面的字符	'f*n'匹配字符 n 前面有任意个字符 f	fn, fan,faan, fabcn
+	匹配前面的字符 1 次或多次	'ba+ '匹配以 b 开头后面紧跟至少有一个 a	ba, bay, bare, battle
<字符串>	匹配包含指定的字符串的文本	'fa'	fan,afa,faad
[字符集合]	匹配字符集合中的任何一个字符	'[xz]' 匹配 x 或者 z	dizzy, zebra, x-ray, extra
[^]	匹配不在括号中的任何字符	'[^abc]'匹配任何不包含 a、b 或 c 的字符串	desk, fox, f8ke
字符串{n,}	匹配前面的字符串至少 n 次	b{2}匹配 2 个或更多的 b	bbb,bbbb,bbbbbbb
字符串{n,m}	匹配前面的字符串至少 n 次,至多 m 次。如果 n 为 0，此参数为可选参数	b{2,4}匹配最少 2 个，最多 4 个 b	bb,bbb,bbbb

下文将详细介绍在 MySQL 中如何使用正则表达式。

7.8.1 查询以特定字符或字符串开头的记录

字符‘^’匹配以特定字符或者字符串开头的文本。

【例 7.68】在 fruits 表中，查询 f_name 字段以字母‘b’开头的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP '^b';
+------+------+--------------+------------+
| f_id | s_id | f_name       | f_price    |
+------+------+--------------+------------+
| b1   |  101 | blackberry   |   10.20    |
| b2   |  104 | berry        |    7.60    |
| t1   |  102 | banana       |   10.30    |
+------+------+--------------+------------+
```

fruits 表中有 3 条记录的 f_name 字段值是以字母 b 开头，返回结果有 3 条记录。

【例 7.69】在 fruits 表中，查询 f_name 字段以“be”开头的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP '^be';
+------+------+--------+---------+
| f_id | s_id | f_name | f_price |
+------+------+--------+---------+
| b2   |  104 | berry  |   7.60  |
+------+------+--------+---------+
```

只有 berry 是以“be”开头，所以查询结果中只有 1 条记录。

7.8.2 查询以特定字符或字符串结尾的记录

字符‘$’匹配以特定字符或者字符串结尾的文本。

【例 7.70】在 fruits 表中，查询 f_name 字段以字母‘y’结尾的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'y$';
+------+------+-----------+---------+
| f_id | s_id | f_name    | f_price |
+------+------+-----------+---------+
| b1   |  101 | blackberry |  10.20 |
| b2   |  104 | berry      |   7.60 |
| c0   |  101 | cherry     |   3.20 |
| m2   |  105 | xbabay     |   2.60 |
+------+------+-----------+---------+
```

fruits 表中有 4 条记录的 f_name 字段值是以字母‘y’结尾，返回结果有 4 条记录。

【例 7.71】在 fruits 表中，查询 f_name 字段以字符串“rry”结尾的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'rry$';
+------+------+-----------+----------+
| f_id | s_id | f_name    | f_price  |
+------+------+-----------+----------+
| b1   |  101 | blackberry |  10.20  |
| b2   |  104 | berry      |   7.60  |
| c0   |  101 | cherry     |   3.20  |
+------+------+-----------+----------+
```

fruits 表中有 3 条记录的 f_name 字段值是以字符串“rry”结尾，返回结果有 3 条记录。

7.8.3 用符号"."来替代字符串中的任意一个字符

字符‘.’匹配任意一个字符。

【例 7.72】在 fruits 表中，查询 f_name 字段值包含字母‘a’与‘g’且两个字母之间只有一个字母的记录，SQL 语句如下，

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'a.g';
+------+------+--------+---------+
| f_id | s_id | f_name | f_price |
+------+------+--------+---------+
| bs1  |  102 | orange | 11.20 |
| m1   |  106 | mango  | 15.70 |
+------+------+--------+---------+
```

查询语句中‘a.g’指定匹配字符中要有字母 a 和 g，且两个字母之间包含单个字符，并不限定匹配的字符的位置和所在查询字符串的总长度，因此 orange 和 mango 都符合匹配条件。

7.8.4 使用"*"和"+"来匹配多个字符

星号‘*’匹配前面的字符任意多次，包括 0 次。加号‘+’匹配前面的字符至少一次。

【例 7.73】在 fruits 表中，查询 f_name 字段值以字母‘b’开头，且‘b’后面出现字母‘a’的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP '^ba*';
+------+------+------------+------------+
| f_id | s_id | f_name     | f_price    |
+------+------+------------+------------+
| b1   |  101 | blackberry |   10.20 |
| b2   |  104 | berry      |    7.60 |
| t1   |  102 | banana     |   10.30 |
+------+------+------------+------------+
```

星号‘*’可以匹配任意多个字符，blackberry 和 berry 中字母 b 后面并没有出现字母 a，但是也满足匹配条件。

【例 7.74】在 fruits 表中，查询 f_name 字段值以字母‘b’开头，且‘b’后面出现字母‘a’至少一次的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP '^ba+';
+------+------+--------+---------+
| f_id | s_id | f_name | f_price |
+------+------+--------+---------+
| t1   |  102 | banana |  10.30 |
+------+------+--------+---------+
```

‘a+’匹配字母‘a’至少一次，只有 banana 满足匹配条件。

7.8.5 匹配指定字符串

正则表达式可以匹配指定字符串，只要这个字符串在查询文本中即可，如要匹配多个字符串，多个字符串之间使用分隔符‘|’隔开。

【例 7.75】在 fruits 表中，查询 f_name 字段值包含字符串“on”的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'on';
+------+------+----------+---------+
| f_id | s_id | f_name   | f_price |
+------+------+----------+---------+
| bs2  |  105 | melon    |  8.20   |
| l2   |  104 | lemon    |  6.40   |
| o2   |  103 | coconut  |  9.20   |
+------+------+----------+---------+
```

可以看到，f_name 字段的 melon、lemon 和 coconut 3 个值中都包含有字符串“on”，满足匹配条件。

【例 7.76】在 fruits 表中，查询 f_name 字段值包含字符串“on”或者“ap”的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'on|ap';
+------+------+----------+---------+
| f_id | s_id | f_name   | f_price |
+------+------+----------+---------+
| a1   |  101 | apple    |  5.20   |
| a2   |  103 | apricot  |  2.20   |
| bs2  |  105 | melon    |  8.20   |
| l2   |  104 | lemon    |  6.40   |
| o2   |  103 | coconut  |  9.20   |
| t2   |  102 | grape    |  5.30   |
+------+------+----------+---------+
```

可以看到，f_name 字段的 melon、lemon 和 coconut 3 个值中都包含有字符串“on”，apple 和 apricot 值中包含字符串“ap”，满足匹配条件。

之前介绍过，LIKE 运算符也可以匹配指定的字符串，但与 REGEXP 不同，LIKE 匹配的字符串如果在文本中间出现，则找不到它，相应的行也不会返回。而 REGEXP 在文本内进行匹配，如果被匹配的字符串在文本中出现，REGEXP 将会找到它，相应的行也会被返回。对比结果如【例 7.77】所示。

【例 7.77】在 fruits 表中，使用 LIKE 运算符查询 f_name 字段值为“on”的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name LIKE 'on';
Empty set (0.00 sec)
```

f_name 字段没有值为“on”的记录，返回结果为空。读者可以体会一下两者的区别。

7.8.6 匹配指定字符中的任意一个

方括号“[]”指定一个字符集合，只匹配其中任何一个字符，即为所查找的文本。

【例 7.78】在 fruits 表中，查找 f_name 字段中包含字母‘o’或者‘t’的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP '[ot]';
+------+------+---------+---------+
| f_id | s_id | f_name  | f_price |
+------+------+---------+---------+
| a2   |  103 | apricot |    2.20 |
| bs1  |  102 | orange  |   11.20 |
| bs2  |  105 | melon   |    8.20 |
| l2   |  104 | lemon   |    6.40 |
| m1   |  106 | mango   |   15.70 |
| m3   |  105 | xxtt    |   11.60 |
| o2   |  103 | coconut |    9.20 |
+------+------+---------+---------+
```

查询结果可以看到，所有返回的记录的 f_name 字段的值中都包含有字母 o 或者 t，或者两个都有。

方括号“[]”还可以指定数值集合。

【例 7.79】在 fruits 表，查询 s_id 字段中数值中包含 4、5 或者 6 的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE s_id REGEXP '[456]';
+------+------+---------+---------+
| f_id | s_id | f_name  | f_price |
+------+------+---------+---------+
| b2   |  104 | berry   |    7.60 |
| bs2  |  105 | melon   |    8.20 |
| l2   |  104 | lemon   |    6.40 |
| m1   |  106 | mango   |   15.70 |
| m2   |  105 | xbabay  |    2.60 |
| m3   |  105 | xxtt    |   11.60 |
+------+------+---------+---------+
```

查询结果中，s_id 字段值中有 3 个数字中的 1 个即为匹配记录字段。

匹配集合“[456]”也可以写成“[4-6]”即指定集合区间。例如“[a-z]”表示集合区间为从 a~z 的字母，“[0-9]”表示集合区间为所有数字。

7.8.7 匹配指定字符以外的字符

“[^字符集合]”匹配不在指定集合中的任何字符。

【例 7.80】在 fruits 表中，查询 f_id 字段包含字母 a~e 和数字 1~2 以外的字符的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_id REGEXP '[^a-e1-2]';
+------+------+---------+---------+
| f_id | s_id | f_name  | f_price |
+------+------+---------+---------+
| b5   |  107 | xxxx    |    3.60 |
| bs1  |  102 | orange  |   11.20 |
| bs2  |  105 | melon   |    8.20 |
| c0   |  101 | cherry  |    3.20 |
| l2   |  104 | lemon   |    6.40 |
| m1   |  106 | mango   |   15.70 |
| m2   |  105 | xbabay  |    2.60 |
| m3   |  105 | xxtt    |   11.60 |
| o2   |  103 | coconut |    9.20 |
| t1   |  102 | banana  |   10.30 |
| t2   |  102 | grape   |    5.30 |
| t4   |  107 | xbababa |    3.60 |
+------+------+---------+---------+
```

返回记录中的 f_id 字段值中包含了指定字母和数字以外的值，如 s、m、o、t 等，这些字母均不在 a~e 与 1~2 之间，满足匹配条件。

7.8.8 使用{n,}或者{n,m}来指定字符串连续出现的次数

“字符串{n,}”表示至少匹配 n 次前面的字符；“字符串{n,m}”表示匹配前面的字符串不少于 n 次，不多于 m 次。例如，a{2,}表示字母 a 连续出现至少 2 次，也可以大于 2 次；a{2,4}表示字母 a 连续出现最少 2 次，最多不能超过 4 次。

【例 7.81】在 fruits 表中，查询 f_name 字段值出现字母‘x’至少 2 次的记录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'x{2,}';
+------+------+--------+---------+
| f_id | s_id | f_name | f_price |
+------+------+--------+---------+
| b5   |  107 | xxxx   |    3.60 |
| m3   |  105 | xxtt   |   11.60 |
+------+------+--------+---------+
```

可以看到，f_name 字段的“xxxx”包含了 4 个字母‘x’，“xxtt”包含两个字母‘x’，均为满足匹配条件的记录。

【例 7.82】在 fruits 表中，查询 f_name 字段值出现字符串“ba”最少 1 次，最多 3 次的记

录，SQL 语句如下：

```
mysql> SELECT * FROM fruits WHERE f_name REGEXP 'ba{1,3}';
+------+---------+----------+---------+
| f_id | s_id    | f_name   | f_price |
+------+---------+----------+---------+
| m2   | 105     | xbabay   |    2.60 |
| t1   | 102     | banana   |   10.30 |
| t4   | 107     | xbababa  |    3.60 |
+------+---------+----------+---------+
```

可以看到，f_name 字段的 xbabay 值中“ba”出现了 2 次，banana 中出现了 1 次，xbababa 中出现了 3 次，都满足匹配条件的记录。

7.9 综合案例——数据表查询操作

SQL 语句可以分为两部分，一部分用来创建数据库对象，另一部分用来操作这些对象，本章详细介绍了操作数据库对象的数据表查询语句。通过本章的介绍，读者可以了解到 SQL 中的查询语言功能的强大，用户可以根据需要灵活使用。本章的综合案例将回顾这些查询语句。

1. 案例目的

根据不同条件对表进行查询操作，掌握数据表的查询语句。employee、dept 表结构以及表中的记录，如表 7.4~表 7.7 所示。

表 7.4 employee 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
e_no	员工编号	INT(11)	是	否	是	是	否
e_name	员工姓名	VARCHAR(50)	否	否	是	否	否
e_gender	员工性别	CHAR(2)	否	否	否	否	否
dept_no	部门编号	INT(11)	否	否	是	否	否
e_job	职位	VARCHAR(50)	否	否	是	否	否
e_salary	薪水	INT(11)	否	否	是	否	否
hireDate	入职日期	DATE	否	否	是	否	否

表 7.5 dept 表结构

字段名	字段说明	数据类型	主键	外键	非空	唯一	自增
d_no	部门编号	INT(11)	是	是	是	是	是
d_name	部门名称	VARCHAR(50)	否	否	是	否	否
d_location	部门地址	VARCHAR(100)	否	否	否	否	否

表 7.6 employee 表中的记录

e_no	e_name	e_gender	dept_no	e_job	e_salary	hireDate
1001	SMITH	m	20	CLERK	800	2005-11-12
1002	ALLEN	f	30	SALESMAN	1600	2003-05-12
1003	WARD	f	30	SALESMAN	1250	2003-05-12
1004	JONES	m	20	MANAGER	2975	1998-05-18
1005	MARTIN	m	30	SALESMAN	1250	2001-06-12
1006	BLAKE	f	30	MANAGER	2850	1997-02-15
1007	CLARK	m	10	MANAGER	2450	2002-09-12
1008	SCOTT	m	20	ANALYST	3000	2003-05-12
1009	KING	f	10	PRESIDENT	5000	1995-01-01
1010	TURNER	f	30	SALESMAN	1500	1997-10-12
1011	ADAMS	m	20	CLERK	1100	1999-10-05
1012	JAMES	f	30	CLERK	950	2008-06-15

表 7.7 dept 表中的记录

d_no	d_name	d_location
10	ACCOUNTING	ShangHai
20	RESEARCH	BeiJing
30	SALES	ShenZhen
40	OPERATIONS	FuJian

2. 案例操作过程

步骤 01 创建数据表 employee 和 dept。

```
CREATE TABLE dept
(
d_no        INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
d_name      VARCHAR(50),NOT NOLL
d_location    VARCHAR(100)
);
```

由于 employee 表 dept_no 依赖于父表 dept 的主键 d_no，因此需要先创建 dept 表，然后创建 employee 表。

```
CREATE TABLE employee
(
e_no        INT(11) NOT NULL PRIMARY KEY,
e_name      VARCHAR(50) NOT NULL,
e_gender    CHAR(2),
dept_no     INT(11) NOT NULL,
e_job       VARCHAR(50) NOT NULL,
e_salary    INT(11) NOT NULL,
```

```
hireDate   DATE  NOT NULL,
CONSTRAINT dno_fk FOREIGN KEY(dept_no)
REFERENCES dept(d_no)
);
```

步骤02 将指定记录分别插入两个表中。

向 dept 表中插入数据，SQL 语句如下：

```
INSERT INTO dept
VALUES (10, 'ACCOUNTING', 'ShangHai'),
(20, 'RESEARCH ', 'BeiJing '),
(30, 'SALES ', 'ShenZhen '),
(40, 'OPERATIONS ', 'FuJian ');
```

向 employee 表中插入数据，SQL 语句如下：

```
INSERT INTO employee
VALUES (1001, 'SMITH', 'm',20, 'CLERK',800,'2005-11-12'),
(1002, 'ALLEN', 'f',30, 'SALESMAN', 1600,'2003-05-12'),
(1003, 'WARD', 'f',30, 'SALESMAN', 1250,'2003-05-12'),
(1004, 'JONES', 'm',20, 'MANAGER', 2975,'1998-05-18'),
(1005, 'MARTIN', 'm',30, 'SALESMAN', 1250,'2001-06-12'),
(1006, 'BLAKE', 'f',30, 'MANAGER', 2850,'1997-02-15'),
(1007, 'CLARK', 'm',10, 'MANAGER', 2450,'2002-09-12'),
(1008, 'SCOTT', 'm',20, 'ANALYST', 3000,'2003-05-12'),
(1009, 'KING', 'f',10, 'PRESIDENT', 5000,'1995-01-01'),
(1010, 'TURNER', 'f',30, 'SALESMAN', 1500,'1997-10-12'),
(1011, 'ADAMS', 'm',20, 'CLERK', 1100,'1999-10-05'),
(1012, 'JAMES', 'm',30, 'CLERK', 950,'2008-06-15');
```

步骤03 在 employee 表中，查询所有记录的 e_no、e_name 和 e_salary 字段值。

```
SELECT e_no, e_name, e_salary;
```

执行结果如下：

```
mysql> SELECT e_no, e_name, e_salary FROM employee;
+------+-------------+------------+
| e_no | e_name      | e_salary   |
+------+-------------+------------+
| 1001 | SMITH       |        800 |
| 1002 | ALLEN       |       1600 |
| 1003 | WARD        |       1250 |
| 1004 | JONES       |       2975 |
| 1005 | MARTIN      |       1250 |
| 1006 | BLAKE       |       2850 |
| 1007 | CLARK       |       2450 |
| 1008 | SCOTT       |       3000 |
```

```
| 1009 | KING   |    5000   |
| 1010 | TURNER |    1500   |
| 1011 | ADAMS  |    1100   |
| 1012 | JAMES  |     950   |
+------+--------+-----------+
12 rows in set (0.00 sec)
```

步骤 04 在 employee 表中，查询 dept_no 等于 10 和 20 的所有记录。

```
SELECT * FROM employee WHERE dept_no IN (10, 20);
```

执行结果如下：

```
mysql> SELECT * FROM employee WHERE dept_no IN (10, 20);
+------+--------+---------+---------+----------+----------+------------+
| e_no | e_name | e_gender | dept_no | e_job    | e_salary | hireDate   |
+------+--------+---------+---------+----------+----------+------------+
| 1001 | SMITH  | m        |      20 | CLERK    |      800 | 2005-11-12 |
| 1004 | JONES  | m        |      20 | MANAGER  |     2975 | 1998-05-18 |
| 1007 | CLARK  | m        |      10 | MANAGER  |     2450 | 2002-09-12 |
| 1008 | SCOTT  | m        |      20 | ANALYST  |     3000 | 2003-05-12 |
| 1009 | KING   | f        |      10 | PRESIDENT|     5000 | 1995-01-01 |
| 1011 | ADAMS  | m        |      20 | CLERK    |     1100 | 1999-10-05 |
+------+--------+---------+---------+----------+----------+------------+
6 rows in set (0.00 sec)
```

步骤 05 在 employee 表中，查询工资范围在 800~2500 的员工信息。

```
SELECT * FROM employee WHERE e_salary BETWEEN 800 AND 2500;
```

执行结果如下：

```
mysql> SELECT * FROM employee WHERE e_salary BETWEEN 800 AND 2500;
+------+--------+---------+---------+----------+----------+------------+
| e_no | e_name | e_gender | dept_no | e_job    | e_salary | hireDat
|
+------+--------+---------+---------+----------+----------+------------+
| 1001 | SMITH  | m        |      20 | CLERK    |      800 | 2005-11-12 |
| 1002 | ALLEN  | f        |      30 | SALESMAN |     1600 | 2003-05-12 |
| 1003 | WARD   | f        |      30 | SALESMAN |     1250 | 2003-05-12 |
| 1005 | MARTIN | m        |      30 | SALESMAN |     1250 | 2001-06-12 |
| 1007 | CLARK  | m        |      10 | MANAGER  |     2450 | 2002-09-12 |
| 1010 | TURNER | f        |      30 | SALESMAN |     1500 | 1997-10-12 |
| 1011 | ADAMS  | m        |      20 | CLERK    |     1100 | 1999-10-05 |
| 1012 | JAMES  | m        |      30 | CLERK    |      950 | 2008-06-15 |
+------+--------+---------+---------+----------+----------+------------+
```

```
-------------+-------------+
8 rows in set (0.00 sec)
```

步骤 06 在 employee 表中，查询部门编号为 20 的部门中的员工信息。

```
SELECT * FROM employee WHERE dept_no = 20;
```

执行结果如下：

```
mysql> SELECT * FROM employee WHERE dept_no = 20;
+------+-------------+-------------+---------------+-------------+--------
--------+-------------+
| e_no | e_name  | e_gender  | dept_no  | e_job    | e_salary   | hireDate    |
+------+-------------+-------------+---------------+-------------+--------
----------+-------------+
| 1001 | SMITH  | m         |      20  | CLERK    |      800   | 2005-11-12 |
| 1004 | JONES  | m         |      20  | MANAGER  |     2975   | 1998-05-18 |
| 1008 | SCOTT  | m         |      20  | ANALYST  |     3000   | 2003-05-12 |
| 1011 | ADAMS  | m         |      20  | CLERK    |     1100   | 1999-10-05 |
+------+-------------+-------------+---------------+-------------+--------
-------------+-------------+
4 rows in set (0.00 sec)
```

步骤 07 在 employee 表中，查询每个部门最高工资的员工信息。

```
SELECT dept_no, MAX(e_salary) FROM employee GROUP BY dept_no;
```

执行结果如下：

```
mysql> SELECT dept_no, MAX(e_salary) FROM employee GROUP BY dept_no;
+---------+-------------------+
| dept_no | MAX(e_salary) |
+---------+-------------------+
|   10    |         5000      |
|   20    |         3000      |
|   30    |         2850      |
+---------+-------------------+
3 rows in set (0.00 sec)
```

步骤 08 查询员工 BLAKE 所在部门和部门所在地。

```
SELECT d_no, d_location  FROM dept WHERE d_no=
(SELECT dept_no FROM employee WHERE e_name='BLAKE');
```

执行结果如下：

```
mysql> SELECT e_name,d_no, d_location
    -> FROM dept WHERE d_no=
    -> (SELECT dept_no FROM employee WHERE e_name='BLAKE');
+--------+-------------+
```

```
| d_no | d_location |
+------+------------+
|  30 | ShenZhen |
+------+------------+
1 row in set (0.00 sec)
```

步骤09 使用连接查询，查询所有员工的部门和部门信息。

```
SELECT e_no, e_name, dept_no, d_name,d_location
FROM employee, dept WHERE dept.d_no=employee.dept_no;
```

执行结果如下：

```
mysql> SELECT e_no, e_name, dept_no, d_name,d_location
    -> FROM employee, dept WHERE dept.d_no=employee.dept_no;
+------+---------+---------+------------+------------+
| e_no | e_name | dept_no |  d_name    | d_location |
+------+---------+---------+------------+------------+
| 1001 | SMITH  |      20 | RESEARCH   | BeiJing    |
| 1002 | ALLEN  |      30 | SALES      | ShenZhen   |
| 1003 | WARD   |      30 | SALES      | ShenZhen   |
| 1004 | JONES  |      20 | RESEARCH   | BeiJing    |
| 1005 | MARTIN |      30 | SALES      | ShenZhen   |
| 1006 | BLAKE  |      30 | SALES      | ShenZhen   |
| 1007 | CLARK  |      10 | ACCOUNTING | ShangHai   |
| 1008 | SCOTT  |      20 | RESEARCH   | BeiJing    |
| 1009 | KING   |      10 | ACCOUNTING | ShangHai   |
| 1010 | TURNER |      30 | SALES      | ShenZhen   |
| 1011 | ADAMS  |      20 | RESEARCH   | BeiJing    |
| 1012 | JAMES  |      30 | SALES      | ShenZhen   |
+------+---------+---------+------------+------------+
12 rows in set (0.00 sec)
```

步骤10 在employee表中，计算每个部门各有多少名员工。

```
SELECT dept_no, COUNT(*) FROM employee GROUP BY dept_no;
```

执行结果如下：

```
mysql> SELECT dept_no, COUNT(*) FROM employee GROUP BY dept_no;
+---------+----------+
| dept_no | COUNT(*) |
+---------+----------+
|      10 |        2 |
|      20 |        4 |
|      30 |        6 |
+---------+----------+
3 rows in set (0.00 sec)
```

步骤 11 在 employee 表中，计算不同类型职工的总工资数。

```
SELECT e_job, SUM(e_salary) FROM employee GROUP BY e_job;
```

执行结果如下：

```
mysql> SELECT e_job, SUM(e_salary) FROM employee GROUP BY e_job;
+-----------+----------------------+
| e_job     | SUM(e_salary)        |
+-----------+----------------------+
| ANALYST   |         3000         |
| CLERK     |         2850         |
| MANAGER   |         8275         |
| PRESIDENT |         5000         |
| SALESMAN  |         5600         |
+-----------+----------------------+
5 rows in set (0.00 sec)
```

步骤 12 在 employee 表中，计算不同部门的平均工资。

```
SELECT dept_no, AVG(e_salary) FROM employee GROUP BY dept_no;
```

执行结果如下：

```
mysql> SELECT dept_no, AVG(e_salary) FROM employee GROUP BY dept_no;
+---------+----------------+
| dept_no | AVG(e_salary) |
+---------+----------------+
|    10   |   3725.0000    |
|    20   |   1968.7500    |
|    30   |   1566.6667    |
+---------+----------------+
3 rows in set (0.00 sec)
```

步骤 13 在 employee 表中，查询工资低于 1500 的员工信息。

```
SELECT * FROM employee WHERE e_salary < 1500;
```

执行过程如下：

```
mysql> SELECT * FROM employee WHERE e_salary < 1500;
+------+--------+----------+---------+---------+---------+------------+
| e_no | e_name | e_gender | dept_no | e_job   | e_salary| hireDate   |
+------+--------+----------+---------+---------+---------+------------+
| 1001 | SMITH  | m        |      20 | CLERK   |     800 | 2005-11-12 |
| 1003 | WARD   | f        |      30 | SALESMAN|    1250 | 2003-05-12 |
| 1005 | MARTIN | m        |      30 | SALESMAN|    1250 | 2001-06-12 |
| 1011 | ADAMS  | m        |      20 | CLERK   |    1100 | 1999-10-05 |
| 1012 | JAMES  | m        |      30 | CLERK   |     950 | 2008-06-15 |
+------+--------+----------+---------+---------+---------+------------+
```

```
5 rows in set (0.00 sec)
```

步骤 14 在 employee 表中，将查询记录先按部门编号由高到低排列，再按员工工资由高到低排列。

```
SELECT e_name,dept_no, e_salary
FROM employee ORDER BY dept_no DESC, e_salary DESC;
```

执行过程如下：

```
mysql> SELECT e_name,dept_no, e_salary
    -> FROM employee ORDER BY dept_no DESC, e_salary DESC;
+-----------+-------------+-----------+
| e_name    | dept_no     | e_salary  |
+-----------+-------------+-----------+
| BLAKE     |          30 |      2850 |
| ALLEN     |          30 |      1600 |
| TURNER    |          30 |      1500 |
| WARD      |          30 |      1250 |
| MARTIN    |          30 |      1250 |
| JAMES     |          30 |       950 |
| SCOTT     |          20 |      3000 |
| JONES     |          20 |      2975 |
| ADAMS     |          20 |      1100 |
| SMITH     |          20 |       800 |
| KING      |          10 |      5000 |
| CLARK     |          10 |      2450 |
+-----------+-------------+-----------+
12 rows in set (0.00 sec)
```

步骤 15 在 employee 表中，查询员工姓名以字母‘A’或‘S’开头的员工的信息。

```
SELECT * FROM employee WHERE e_name REGEXP '^[as]';
```

执行过程如下：

```
mysql> SELECT * FROM employee WHERE e_name REGEXP '^[as]';
+------+-----------+-----------+-------------+-----------+-----------+------------+
| e_no | e_name    | e_gender  | dept_no     | e_job     | e_salary  | hireDate   |
+------+-----------+-----------+-------------+-----------+-----------+------------+
| 1001 | SMITH     | m         |          20 | CLERK     |       800 | 2005-11-12 |
| 1002 | ALLEN     | f         |          30 | SALESMAN  |      1600 | 2003-05-12 |
| 1008 | SCOTT     | m         |          20 | ANALYST   |      3000 | 2003-05-12 |
| 1011 | ADAMS     | m         |          20 | CLERK     |      1100 | 1999-10-05 |
+------+-----------+-----------+-------------+-----------+-----------+------------+
```

```
4 rows in set (0.00 sec)
```

步骤 16 在 employee 表中，查询到目前为止，工龄大于等于 10 年的员工信息。

```
SELECT * FROM employee where YEAR(CURDATE()) -YEAR(hireDate) >= 15;
```

执行过程如下：

```
mysql> SELECT * FROM employee where YEAR(CURDATE()) -YEAR(hireDate) >= 15;
+------+-----------+----------+---------+-----------+----------+------------+
| e_no | e_name    | e_gender | dept_no | e_job     | e_salary | hireDate   |
+------+-----------+----------+---------+-----------+----------+------------+
| 1004 | JONES     | m        |      20 | MANAGER   |     2975 | 1998-05-18 |
| 1005 | MARTIN    | m        |      30 | SALESMAN  |     1250 | 2001-06-12 |
| 1006 | BLAKE     | f        |      30 | MANAGER   |     2850 | 1997-02-15 |
| 1009 | KING      | f        |      10 | PRESIDENT |     5000 | 1995-01-01 |
| 1010 | TURNER    | f        |      30 | SALESMAN  |     1500 | 1997-10-12 |
| 1011 | ADAMS     | m        |      20 | CLERK     |     1100 | 1999-10-05 |
+------+-----------+----------+---------+-----------+----------+------------+
6 rows in set (0.01 sec)
```

7.10 专家解惑

疑问 1：DISTINCT 可以应用于所有的列吗？

查询结果中，如果需要对列进行降序排序，可以使用 DESC，这个关键字只能对其前面的列进行降序排列。例如，要对多列都进行降序排序，必须要在每一列的列名后面加 DESC 关键字。而 DISTINCT 不同，DISTINCT 不能部分使用。换句话说，DISTINCT 关键字应用于所有列而不仅是它后面的第一个指定列。例如，查询 3 个字段 s_id，f_name，f_price，如果不同记录的这 3 个字段的组合值都不同，则所有记录都会被查询出来。

疑问 2：ORDER BY 可以和 LIMIT 混合使用吗？

在使用 ORDER BY 子句时，应保证其位于 FROM 子句之后，如果使用 LIMIT，则必须位于 ORDER BY 之后，如果子句顺序不正确，MySQL 将产生错误消息。

疑问 3：什么时候使用引号？

在查询的时候，会看到在 WHERE 子句中使用条件，有的值加上了单引号，而有的值未

加。单引号用来限定字符串，如果将值与字符串类型列进行比较，则需要限定引号；而用来与数值进行比较则不需要用引号。

疑问 4：在 WHERE 子句中必须使用圆括号吗？

任何时候使用具有 AND 和 OR 操作符的 WHERE 子句，都应该使用圆括号明确操作顺序。如果条件较多，即使能确定计算次序，默认的计算次序也可能会使 SQL 语句不易理解，因此使用括号明确操作符的次序，是一个良好的习惯。

疑问 5：为什么使用通配符格式正确，却没有查找出符合条件的记录？

MySQL 中存储字符串数据时，可能会不小心把两端带有空格的字符串保存到记录中，而在查看表中记录时，MySQL 不能明确地显示空格，数据库操作者不能直观地确定字符串两端是否有空格。例如，使用 LIKE '%e'匹配以字母 e 结尾的水果的名称，如果字母 e 后面多了一个空格，则 LIKE 语句不能将该记录查找出来。解决的方法是使用 TRIM 函数，将字符串两端的空格删除之后再进行匹配。

7.11 经典习题

在已经创建的 employee 表中进行如下操作：

（1）计算所有女员工（'F'）的年龄。

（2）使用 LIMIT 查询从第 3 条记录开始到第 6 条记录。

（3）查询销售人员（SALSEMAN）的最低工资。

（4）查询名字以字母 N 或者 S 结尾的记录。

（5）查询在 Beijing 工作的员工的姓名和职务。

（6）使用左连接方式查询 employee 和 dept 表。

（7）查询所有 2001~2005 年入职的员工的信息，查询部门编号为 20 和 30 的员工信息并使用 UNION 合并两个查询结果。

（8）使用 LIKE 查询员工姓名中包含字母 a 的记录。

（9）使用 REGEXP 查询员工姓名中包含 T、C 或者 M 3 个字母中任意 1 个的记录。