MySQL 必知必会

LaTeX by Red Bamboo
Jilin University

fall 2024

目录

1 数据库基础

Defination 1

1. **数据库（database）**保存有组织的数据的容器（通常是一个文件或一组文件）
2. **表（table）**某种特定类型数据的结构化清单。
3. **模式（schema）**关于数据库和表的布局及特性的信息。
4. **列（column）**表中的一个字段。所有表都是由一个或多个列组成的。
5. **数据类型（datatype）**所容许的数据的类型。每个表列都有相应的数据类型，它限制（或容许）该列中存储的数据。
6. **行（row）**表中的一个记录。
7. **主键（primary key）**一列（或一组列），其值能够唯一区分表中每个行。

2 连接 MySQL

在操作系统命令提示符下输入 mysql，可以输入以下指令：

```
mysql -u hzz -p
```

Listing 1: 连接 mysql

然后在接下来的对话框中输入密码即可，输入密码完成后会返回如下：

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38
Server version: 8.0.39 MySQL Community Server - GPL
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Listing 2: 连接 mysql 后的回复

3 使用 MySQL

3.1 选择数据库

首先需要创建一个数据库：

```
create crashcourse;
```

Listing 3: 创建数据库

然后调用这个数据库：

```
use sakila;
```

Listing 4: 调用数据库

若使用成功会返回：

```
Database changed
```

Listing 5: 调用数据库返回值

3.2 展示数据和表

展示数据库可以用如下命令:

```
show databases;
```

Listing 6: 显示数据库

此时会返回可用数据库的一个列表:

```
+--------------------+
| Database           |
+--------------------+
| crashcourse        |
| demo               |
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
8 rows in set (0.00 sec)
```

Listing 7: 数据库列表

查看一个数据库中的表可以用如下命令:

```
show tables;
```

Listing 8: 显示数据库

此时会返回可用表的一个列表:

```
+---------------------------+
| Tables_in_sakila          |
+---------------------------+
| actor                     |
| actor_info                |
| address                   |
| category                  |
| city                      |
| country                   |
| customer                  |
| customer_list             |
| film                      |
| film_actor                |
| film_category             |
| film_list                 |
| film_text                 |
| inventory                 |
| language                  |
| nicer_but_slower_film_list |
| payment                   |
| rental                    |
| sales_by_film_category    |
| sales_by_store            |
| staff                     |
| staff_list                |
| store                     |
+---------------------------+
23 rows in set (0.00 sec)
```

Listing 9: 表列表

SHOW 也可以用来显示表列：

```
show columns from actor;
```

Listing 10: 显示列

此时会返回可用列的一个列表：

```
+-------------+------------------+------+-----+-------------------+-------------------------------
| Field       | Type             | Null | Key | Default           | Extra
+-------------+------------------+------+-----+-------------------+-------------------------------
| actor_id    | smallint unsigned | NO   | PRI | NULL              | auto_increment
| first_name  | varchar(45)      | NO   |     | NULL              |
| last_name   | varchar(45)      | NO   | MUL | NULL              |
| last_update | timestamp        | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURREN
+-------------+------------------+------+-----+-------------------+-------------------------------
4 rows in set (0.00 sec)
```

Listing 11: 表列

注意，describe 与 show columns 是等价命令，也就是如下命令也可以返回上面的结果：

```
describe actor;
```

Listing 12: describe

show 还有其他语句：

```
show status;                    # 用于显示广泛的服务器状态信息；
show grants;                    # 用来显示授予用户（所有用户或特定用户）的安全权限；
show errors;                    # 用来显示服务器错误消息。
show warnings;                  # 用来显示服务器警告消息。
```

Listing 13: show 的其他语句

4 检索数据

4.1 检索单个列

首先是先检索表的单列：

```
select name from language;
```

Listing 14: select 单列

输出的结果为：

```
+----------+
| name     |
+----------+
| English  |
| Italian  |
| Japanese |
| Mandarin |
| French   |
| German   |
+----------+
6 rows in set (0.00 sec)
```

Listing 15: select 单列结果

Tips 4.1

1. 在没有排序的情况下，这个数据顺序没有任何意义。
2. 多条 SQL 语句必须以分号（;）分隔。MySQL 不需要在单条 SQL 语句后加分号。(但尽可能加上分号)
3. SQL 语句不区分大小写（可以对所有 SQL 关键字使用大写，而对所有列和表名使用小写）。
4. 在处理 SQL 语句时，其中所有空格都被忽略。

4.2 检索多个列

从一个表中检索多个列必须在 SELECT 关键字后给出多个列名，列名之间必须以逗号分隔:

```
select language_id,name,last_update from language;
```

Listing 16: select 多列

输出的结果为:

```
+-------------+----------+---------------------+
| language_id | name     | last_update         |
+-------------+----------+---------------------+
|           1 | English  | 2006-02-15 05:02:19 |
|           2 | Italian  | 2006-02-15 05:02:19 |
|           3 | Japanese | 2006-02-15 05:02:19 |
|           4 | Mandarin | 2006-02-15 05:02:19 |
|           5 | French   | 2006-02-15 05:02:19 |
|           6 | German   | 2006-02-15 05:02:19 |
+-------------+----------+---------------------+
6 rows in set (0.00 sec)
```

Listing 17: select 多列结果

Tips 4.2

1. 选择多个列时，一定要在列名之间加上逗号，但最后一个列名后不加。
2. SQL 语句一般返回原始的、无格式的数据。数据的格式化是一个表示问题，而不是一个检索问题。

4.3 检索所有列

SELECT 语句检索所有的列可以通过星号（*）通配符来达到，如下所示:

```
SELECT * FROM store;
```

Listing 18: select 所有列

输出的结果为:

```
+----------+-----------------+------------+---------------------+
| store_id | manager_staff_id | address_id | last_update         |
+----------+-----------------+------------+---------------------+
|        1 |               1 |          1 | 2006-02-15 04:57:12 |
|        2 |               2 |          2 | 2006-02-15 04:57:12 |
+----------+-----------------+------------+---------------------+
2 rows in set (0.00 sec)
```

Listing 19: select 所有列结果

Tips 4.3

1. 虽然使用通配符可能会使你自己省事，但检索不需要的列通常会降低检索和应用程序的性能。
2. 使用通配符能检索出名字未知的列。

4.4 检索不同的行（DISTINCT）

只返回不同的值可以用 DISTINCT, 如以下所示：

```
SELECT DISTINCT staff_id FROM rental;
```

Listing 20: select 不同行

输出的结果为：

```
+----------+
| staff_id |
+----------+
|        1 |
|        2 |
+----------+
2 rows in set (0.00 sec)
```

Listing 21: select 所有列结果

但如果直接 SELECT，不加 DISTINCT 就会有 16044 条。

Tips 4.4

不能部分使用 DISTINCT ,DISTINCT 关键字应用于所有列而不仅是前置它的列。

4.5 限制结果（LIMIT）

为了返回第一行或前几行，可使用 LIMIT 子句：

```
SELECT actor_id FROM actor_info LIMIT 5;
```

Listing 22: LIMIT 子句

输出的结果为：

```
+----------+
| actor_id |
+----------+
|        1 |
|        2 |
|        3 |
|        4 |
|        5 |
+----------+
5 rows in set (1.15 sec)
```

Listing 23: LIMIT 的结果

这个结果就是返回开始的 5 行
如果要返回从第 9 行开始的 5 行，可以执行以下命令：

```
SELECT actor_id FROM actor_info LIMIT 9,5;
```

Listing 24: 从特定行开始的 LIMIT 子句

输出的结果为：

```
+----------+
| actor_id |
+----------+
|       10 |
|       11 |
|       12 |
|       13 |
|       14 |
+----------+
5 rows in set (1.20 sec)
```

Listing 25: 从特定行开始的 LIMIT 的结果

Tips 4.5

1. 检索出来的第一行为行 0 而不是行 1。因此，LIMIT 1, 1 将检索出第二行而不是第一行。
2. 在行数不够时，MySQL 将只返回它能返回的那么多行。

4.6 使用完全限定的表名

可以使用限定列列名来索引列，以下两条命令与第一条本章第一条的结果完全相同：

```
SELECT language.name FROM language;
SELECT language.name FROM sakila.language;
```

Listing 26: 限定表名

5 排序数据

5.1 普通排序

之前 SQL 语句返回数据库表的单列，检索出的数据并不是随机显示的，而是以它在底层表中出现的顺序显示。但是，如果数据后来进行过更新或删除，则此顺序将会受到 MySQL 重用回收存储空间的影响。

Defination 5.1 子句（clause）

SQL 语句由子句构成，有些子句是必需的，而有的是可选的。一个子句通常由一个关键字和所提供的数据组成。

因此，为了排序 SELECT 语句检索出来的数据，我们可以使用 ORDER BY 语句：

```
SELECT city FROM city ORDER BY city_id LIMIT 10;
```

Listing 27: ORDER BY 子句

输出的结果为：

```
+----------------------+
| city                 |
+----------------------+
| A Coru?a (La Coru?a) |
| Abha                 |
| Abu Dhabi            |
| Acu?a                |
| Adana                |
| Addis Abeba          |
| Aden                 |
| Adoni                |
| Ahmadnagar           |
| Akishima             |
+----------------------+
10 rows in set (0.00 sec)
```

Listing 28: ORDER BY 子句的结果

Tips 5.1

通常 ORDER BY 子句中使用的列是为显示所选择的列。但是可以通过非选择列进行排序。

5.2 按多列排序

按多个列排序，只要指定列名，列名之间用逗号分开即可：

```
SELECT last_name,first_name,store_id FROM customer ORDER BY last_name,first_name LIMIT 10;
```

Listing 29: 多列排序

输出的结果为：

```
+-----------+------------+----------+
| last_name | first_name | store_id |
+-----------+------------+----------+
| ABNEY     | RAFAEL     |        1 |
| ADAM      | NATHANIEL  |        1 |
| ADAMS     | KATHLEEN   |        2 |
| ALEXANDER | DIANA      |        1 |
| ALLARD    | GORDON     |        1 |
| ALLEN     | SHIRLEY    |        2 |
| ALVAREZ   | CHARLENE   |        2 |
| ANDERSON  | LISA       |        2 |
| ANDREW    | JOSE       |        1 |
| ANDREWS   | IDA        |        2 |
+-----------+------------+----------+
10 rows in set (0.00 sec)
```

Listing 30: 多列排序的结果

5.3 指定排序方向

Tips 5.3

1. 默认排序是升序排序（从 A 到 Z）。
2. 降序排序（从 Z 到 A）必须指定 DESC 关键字。
3. 在多个列上降序排序，必须对每个列指定 DESC 关键字。（与 DESC 对应的是 ASC）
4. 在字典（dictionary）排序顺序中，A 被视为与 a 相同，这是 MySQL（和大多数数据库管理系统）的默认行为。（也就是不区分 A 和 a）。
5. ORDER BY 子句位于 FROM 子句之后，LIMIT 位于 ORDER BY 之后。

降序排列的 SQL 语句如下所示：

```
SELECT last_name,first_name,last_update FROM actor ORDER BY last_name DESC LIMIT 10;
```

Listing 31: 降序排列

输出的结果为：

```
+-------------+------------+---------------------+
| last_name   | first_name | last_update         |
+-------------+------------+---------------------+
| ZELLWEGER   | JULIA      | 2006-02-15 04:34:33 |
| ZELLWEGER   | CAMERON    | 2006-02-15 04:34:33 |
| ZELLWEGER   | MINNIE     | 2006-02-15 04:34:33 |
| WRAY        | CAMERON    | 2006-02-15 04:34:33 |
| WOOD        | FAY        | 2006-02-15 04:34:33 |
| WOOD        | UMA        | 2006-02-15 04:34:33 |
| WITHERSPOON | ANGELA     | 2006-02-15 04:34:33 |
| WINSLET     | FAY        | 2006-02-15 04:34:33 |
| WINSLET     | RIP        | 2006-02-15 04:34:33 |
| WILSON      | WILL       | 2006-02-15 04:34:33 |
+-------------+------------+---------------------+
10 rows in set (0.00 sec)
```

Listing 32: 降序排列的结果

如果多列降序排列，可以参照以下 SQL 语句：

```
SELECT amount,rental_id,staff_id FROM payment ORDER BY amount DESC,rental_id LIMIT 10;
```

Listing 33: 多列降序排列

输出的结果为：

```
+--------+-----------+----------+
| amount | rental_id | staff_id |
+--------+-----------+----------+
|  11.99 |       106 |        2 |
|  11.99 |      2166 |        1 |
|  11.99 |      3973 |        1 |
|  11.99 |      4383 |        2 |
|  11.99 |      8831 |        2 |
|  11.99 |     11479 |        2 |
|  11.99 |     14759 |        1 |
|  11.99 |     14763 |        2 |
|  11.99 |     15415 |        2 |
|  11.99 |     16040 |        2 |
+--------+-----------+----------+
10 rows in set (0.00 sec)
```

Listing 34: 多列降序排序的结果

这种方法（ORDER BY 和 LIMIT 的组合）的一个变种就是直接可以检索出极大值或者极小值，由于上面是极大值，下面我以极小值举例子：

```
SELECT amount FROM payment ORDER BY amount LIMIT 1;
```

Listing 35: 检索极值

输出的结果为：

```
+--------+
| amount |
+--------+
|   0.00 |
+--------+
1 row in set (0.00 sec)
```

Listing 36: 检索极值的结果

6 过滤数据

6.1 WHERE 子句

Defination 6.1 搜索条件（search criteria）

搜索条件也称为过滤条件（filter condition），只检索所需数据需要指定搜索条件（search criteria）。

SELECT 语句中，数据根据 WHERE 子句中指定的搜索条件进行过滤，命令如下：

```
SELECT customer_id,staff_id,amount FROM payment WHERE staff_id = 1 LIMIT 10;
```

Listing 37: WHERE 子句

输出的结果为：

```
+-------------+----------+--------+
| customer_id | staff_id | amount |
+-------------+----------+--------+
|           1 |        1 |   2.99 |
|           1 |        1 |   0.99 |
|           1 |        1 |   5.99 |
|           1 |        1 |   4.99 |
|           1 |        1 |   4.99 |
|           1 |        1 |   3.99 |
|           1 |        1 |   5.99 |
|           1 |        1 |   4.99 |
|           1 |        1 |   4.99 |
|           1 |        1 |   7.99 |
+-------------+----------+--------+
10 rows in set (0.00 sec)
```

Listing 38: WHERE 子句的结果

Tips 6.1

1. 同时使用 ORDER BY 和 WHERE 子句时,ORDER BY 位于 WHERE 之后。
2. WHERE 子句在表名（FROM 子句）之后。

3. 单引号用来限定字符串。如果将值与串类型的列进行比较，则需要限定引号。用来与数值列进行比较的值不用引号。

4. NULL 具有特殊含义，在过滤使用不匹配（<>,！=）过滤选择出不具有特定值的行的时候，并不会返回空值，因此需要空值查询。

表 1: **WHERE 子句操作符**

操作符	说明
=	等于
<>	不等于
！=	不等于
<	小于
<=	小于等于
>	大于
>=	大于等于
BETWEEN	在指定的两个值之间

6.2 检查单个值

首先是用等号的 WHERE 子句：

```
SELECT customer_id,email,address_id FROM customer WHERE customer_id = 7;
```

Listing 39: 用等号的 WHERE 子句

输出的结果为：

```
+-------------+----------------------------------+------------+
| customer_id | email                            | address_id |
+-------------+----------------------------------+------------+
|           7 | MARIA.MILLER@sakilacustomer.org |         11 |
+-------------+----------------------------------+------------+
1 row in set (0.00 sec)
```

Listing 40: 用等号的 WHERE 子句的结果

接着可以用小于、小于等于、大于、大于等于的 WHERE 子句进行搜索：

```
SELECT amount,payment_date,last_update FROM payment
WHERE amount > 11.00
ORDER BY amount DESC
LIMIT 5;
```

Listing 41: 用大于的 WHERE 子句

输出的结果为：

```
+--------+---------------------+---------------------+
| amount | payment_date        | last_update         |
+--------+---------------------+---------------------+
|  11.99 | 2005-07-29 22:37:41 | 2006-02-15 22:12:31 |
|  11.99 | 2005-08-21 23:34:00 | 2006-02-15 22:13:08 |
|  11.99 | 2005-08-23 22:19:33 | 2006-02-15 22:14:00 |
|  11.99 | 2005-05-25 18:18:19 | 2006-02-15 22:14:00 |
|  11.99 | 2005-08-22 23:48:56 | 2006-02-15 22:14:07 |
+--------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

Listing 42: 用大于的 WHERE 子句的结果

6.3 不匹配检查

如下是采用不匹配检查的 SQL 语句，其中 <> 与！= 等价：

```
SELECT name,last_update FROM language WHERE name <> 'Japanese';
```

Listing 43: 用不匹配检查的 WHERE 子句

输出的结果为：

```
+----------+---------------------+
| name     | last_update         |
+----------+---------------------+
| English  | 2006-02-15 05:02:19 |
| Italian  | 2006-02-15 05:02:19 |
| Mandarin | 2006-02-15 05:02:19 |
| French   | 2006-02-15 05:02:19 |
| German   | 2006-02-15 05:02:19 |
+----------+---------------------+
5 rows in set (0.00 sec)
```

Listing 44: 用不匹配检查的 WHERE 子句的结果

6.4 范围值检查

范围值检查需要用 BETWEEN，如下是使用 BETWEEN 的 SQL 语句：

```
SELECT amount,payment_date,last_update FROM payment
WHERE amount BETWEEN 10 AND 11
ORDER BY amount DESC
LIMIT 5;
```

Listing 45: 用 BETWEEN 的 WHERE 子句

输出的结果为：

```
+--------+---------------------+---------------------+
| amount | payment_date        | last_update         |
+--------+---------------------+---------------------+
|  10.99 | 2005-07-30 13:47:43 | 2006-02-15 22:12:30 |
|  10.99 | 2005-07-27 20:23:12 | 2006-02-15 22:12:30 |
|  10.99 | 2005-08-01 06:50:26 | 2006-02-15 22:12:31 |
|  10.99 | 2005-06-21 01:04:35 | 2006-02-15 22:12:33 |
|  10.99 | 2005-07-09 21:55:19 | 2006-02-15 22:12:35 |
+--------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

Listing 46: 用 BETWEEN 的 WHERE 子句的结果

6.5 空值检查

Defination 6.5 NULL 无值（no value）

在一个列不包含值时，称其为包含空值 NULL，它与字段包含 0、空字符串或仅仅包含空格不同。

空值检查的 SQL 语句如下：

```
SELECT rental_date,return_date,staff_id
FROM rental
WHERE return_date IS NULL
LIMIT 5;
```

Listing 47: 空值检查的 WHERE 子句

输出的结果为：

```
+---------------------+-------------+----------+
| rental_date         | return_date | staff_id |
+---------------------+-------------+----------+
| 2006-02-14 15:16:03 | NULL        |        1 |
| 2006-02-14 15:16:03 | NULL        |        1 |
| 2006-02-14 15:16:03 | NULL        |        1 |
| 2006-02-14 15:16:03 | NULL        |        2 |
| 2006-02-14 15:16:03 | NULL        |        1 |
+---------------------+-------------+----------+
5 rows in set (0.00 sec)
```

Listing 48: 空值检查的 WHERE 子句的结果

7 数据过滤

7.1 操作符

Defination 7.1 操作符（operator）

用来联结或改变 WHERE 子句中的子句的关键字。也称为逻辑操作符（logical operator）。

7.2 AND 操作符

AND 操作符给 WHERE 子句附加条件, 参考以下 SQL 语句:

```
SELECT amount,customer_id,staff_id
FROM payment
WHERE amount BETWEEN 10 AND 11
AND staff_id = 1
AND customer_id BETWEEN 0 AND 100;
```

Listing 49: 含 AND 操作符 WHERE 子句

输出的结果为:

```
+--------+-------------+----------+
| amount | customer_id | staff_id |
+--------+-------------+----------+
|  10.99 |          21 |        1 |
|  10.99 |          33 |        1 |
|  10.99 |          45 |        1 |
|  10.99 |          54 |        1 |
|  10.99 |          76 |        1 |
|  10.99 |          78 |        1 |
|  10.99 |          86 |        1 |
+--------+-------------+----------+
7 rows in set (0.01 sec)
```

Listing 50: 含 AND 操作符 WHERE 子句的结果

Defination 7.2 AND 操作符

用在 WHERE 子句中的关键字，用来指示检索满足所有给定条件的行。

7.3 OR 操作符

OR 操作符指示 MySQL 检索匹配任一条件的行，SQL 命令如下所示:

```
SELECT payment_date,rental_id,amount
FROM payment
WHERE amount = 0.99
OR rental_id = 76
LIMIT 10;
```

Listing 51: 含 OR 操作符 WHERE 子句

输出的结果为:

```
+---------------------+-----------+--------+
| payment_date        | rental_id | amount |
+---------------------+-----------+--------+
| 2005-05-25 11:30:37 |        76 |   2.99 |
| 2005-05-28 10:35:23 |       573 |   0.99 |
| 2005-06-15 18:02:53 |      1422 |   0.99 |
| 2005-06-18 13:33:59 |      2363 |   0.99 |
| 2005-07-28 17:33:39 |      8074 |   0.99 |
| 2005-07-28 19:20:07 |      8116 |   0.99 |
| 2005-08-02 18:01:38 |     11367 |   0.99 |
| 2005-08-18 03:57:29 |     12250 |   0.99 |
| 2005-08-19 09:55:16 |     13068 |   0.99 |
| 2005-08-21 23:33:57 |     14762 |   0.99 |
+---------------------+-----------+--------+
10 rows in set (0.00 sec)
```

Listing 52: 含 OR 操作符 WHERE 子句的结果

7.4 计算次序

Tips 7.4

1. 同时有 AND 和 OR 的 WHERE 子句，优先处理 AND，因为 AND 在计算次序中优先级更高
2. 如果需要改变优先级，需要圆括号明确地分组操作符。(即使默认次序是正确的情况下)

例如，我要查询 actor_id 为 1 的并且 film_id 为 166 或 277 的记录，如果写成以下这样：

```
SELECT * FROM film_actor
WHERE actor_id =1 AND film_id = 166
OR film_id = 277;
```

Listing 53: 错误次序的 WHERE 子句

输出的结果为：

```
+----------+---------+---------------------+
| actor_id | film_id | last_update         |
+----------+---------+---------------------+
|        1 |     166 | 2006-02-15 05:05:03 |
|        1 |     277 | 2006-02-15 05:05:03 |
|       24 |     277 | 2006-02-15 05:05:03 |
|       37 |     277 | 2006-02-15 05:05:03 |
|      107 |     277 | 2006-02-15 05:05:03 |
|      115 |     277 | 2006-02-15 05:05:03 |
+----------+---------+---------------------+
6 rows in set (0.00 sec)
```

Listing 54: 错误次序的 WHERE 子句的结果

此时显然是不对的，我查询成了，actor_id 为 1 且 film_id 为 166 或者 film_id 为 277 的情况，因此在这种情况下应该将语句改为：

```
SELECT * FROM film_actor
WHERE actor_id =1
AND (film_id = 166 OR film_id = 277);
```

Listing 55: 正确次序的 WHERE 子句

输出的结果为：

```
+----------+---------+---------------------+
| actor_id | film_id | last_update         |
+----------+---------+---------------------+
|        1 |     166 | 2006-02-15 05:05:03 |
|        1 |     277 | 2006-02-15 05:05:03 |
+----------+---------+---------------------+
2 rows in set (0.00 sec)
```

Listing 56: 正确次序的 WHERE 子句的结果

7.5 IN 操作符

IN 操作符用来指定条件范围，范围中的每个条件都可以进行匹配，因此 IN 操作符在一定程度上可以代替 OR 操作符，我们仍然用上面那个例子：

```
SELECT * FROM film_actor
WHERE actor_id =1
AND film_id IN (166,277);
```

Listing 57: 含 OR 操作符 WHERE 子句

输出的结果为：

```
+----------+---------+---------------------+
| actor_id | film_id | last_update         |
+----------+---------+---------------------+
|        1 |     166 | 2006-02-15 05:05:03 |
|        1 |     277 | 2006-02-15 05:05:03 |
+----------+---------+---------------------+
2 rows in set (0.00 sec)
```

Listing 58: 含 OR 操作符 WHERE 子句的结果

由此可以给出 IN 操作符的定义：

Defination 7.5 IN 操作符

WHERE 子句中用来指定要匹配值的清单的关键字，功能与 OR 相当。

Tips 7.5

1. IN 操作符的语法更清楚且更直观。
2. IN 操作符计算的次序更容易管理。

3. IN 操作符一般比 OR 操作符清单执行更快。

4. IN 操作符可以包含其他 SELECT 语句，使得能够更动态地建立 WHERE 子句。

7.6 NOT 操作符

Defination 7.6 NOT 操作符

WHERE 子句中用来否定后跟条件的关键字。

这个例子改成不是 actor_id 为 1 且 film_id 为 166 或者 film_id 为 277，并且限制 10 条，则有：

```
SELECT * FROM film_actor
WHERE actor_id =1
AND film_id NOT IN (166,277)
LIMIT 10;
```

Listing 59: 含 NOT 操作符 WHERE 子句

输出的结果为：

```
+----------+---------+---------------------+
| actor_id | film_id | last_update         |
+----------+---------+---------------------+
|        1 |       1 | 2006-02-15 05:05:03 |
|        1 |      23 | 2006-02-15 05:05:03 |
|        1 |      25 | 2006-02-15 05:05:03 |
|        1 |     106 | 2006-02-15 05:05:03 |
|        1 |     140 | 2006-02-15 05:05:03 |
|        1 |     361 | 2006-02-15 05:05:03 |
|        1 |     438 | 2006-02-15 05:05:03 |
|        1 |     499 | 2006-02-15 05:05:03 |
|        1 |     506 | 2006-02-15 05:05:03 |
|        1 |     509 | 2006-02-15 05:05:03 |
+----------+---------+---------------------+
10 rows in set (0.00 sec)
```

Listing 60: 含 NOT 操作符 WHERE 子句的结果

Tips 7.6

MySQL 支持使用 NOT 对 IN 、BETWEEN 和 EXISTS 子句取反。

8 通配符

8.1 LIKE 操作符

Defination 8.1 通配符（wildcard）

用来匹配值的一部分的特殊字符。

Defination 8.1 搜索模式（search pattern）

由字面值、通配符或两者组合构成的搜索条件。

在搜索子句中使用通配符必须使用 LIKE 操作符。LIKE 指示 MySQL，后跟的搜索模式利用通配符匹配而不是直接相等匹配进行比较。

Tips 8.1

1. LIKE 是谓词而不是操作符。
2. 操作符作为谓词（predicate）时不是操作符。

8.2 百分号（%）通配符

Defination 8.2 百分号（%）通配符

% 表示任何字符出现任意次数。

例如，我要找到 address 表中 district 字段所有开头 Ha 的记录，就可以采取以下 SQL 语句：

```
SELECT DISTINCT district FROM address WHERE district LIKE 'Ha%';
```

Listing 61: 百分号通配符

输出的结果为：

```
+----------+
| district |
+----------+
| Hamilton |
| Haryana  |
| Ha Darom |
| Haiphong |
| Haskovo  |
| Hawalli  |
| Hanoi    |
| Hainan   |
+----------+
8 rows in set (0.00 sec)
```

Listing 62: 含百分号通配符的结果

Tips 8.2

1. **区分大小写** 根据 MySQL 的配置方式，搜索可以是区分大小写的。
2. **注意尾空格** 尾空格可能会干扰通配符匹配。
3. **注意 NULL** 通配符不可以匹配 NULL。
4. **匹配字符数** % 代表搜索模式中给定位置的 0 个、1 个或多个字符。

8.3 下划线（_）通配符

Defination 8.3 下划线（_）通配符

下划线 _ 的用途与% 一样，但下划线只匹配单个字符而不是多个字符。

现在我要找到 customer 中字段名为 first_name 中包含开头为 ERI 的数据，则 SQL 语句可以是：

```
SELECT first_name,last_name FROM customer WHERE first_name LIKE 'ERI_';
```

Listing 63: 下划线通配符

输出的结果为：

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ERIN       | DUNN      |
| ERIC       | ROBERT    |
| ERIK       | GUILLEN   |
+------------+-----------+
3 rows in set (0.00 sec)
```

Listing 64: 含下划线通配符的结果

9 正则表达式

9.1 基本字符匹配

REGEXP 跟的东西为正则表达式，我们查询 first_name 字段包含 ERI 的所有记录，SQL 语句可以为：

```
SELECT first_name,last_name FROM customer WHERE first_name REGEXP 'ERI';
```

Listing 65: 基本字符匹配

输出的结果为：

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| CATHERINE  | CAMPBELL  |
| KATHERINE  | RIVERA    |
| VALERIE    | BLACK     |
| ERIN       | DUNN      |
| ERICA      | MATTHEWS  |
| ERIKA      | PENA      |
| ERIC       | ROBERT    |
| FREDERICK  | ISBELL    |
| ERIK       | GUILLEN   |
+------------+-----------+
9 rows in set (0.00 sec)
```

Listing 66: 基本字符匹配的结果

正则表达式中表 示为任意一个字符，这点可以实现与 LIKE 相同的效果，可以查找 customer 表中 id 为 177，277 ……等记录：

```
SELECT customer_id,first_name,last_name FROM customer WHERE customer_id REGEXP '.77';
```

Listing 67: 匹配任意一个字符的正则表达式字符

输出的结果为：

```
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         177 | SAMANTHA   | DUNCAN    |
|         277 | OLGA       | JIMENEZ   |
|         377 | HOWARD     | FORTNER   |
|         477 | DAN        | PAINE     |
|         577 | CLIFTON    | MALCOLM   |
+-------------+------------+-----------+
```

Listing 68: 含匹配任意一个字符的正则表达式字符的结果

Tips 9.1.1 LIKE 与 RECEGP 的区别

1. LIKE 匹配整个列。LIKE 不会在列值找到被匹配的文本相应的行也不被返回。
2. REGEXP 在列值内进行匹配，REGEXP 会找到在列值中被匹配的文本。

Tips 9.1.2 匹配不区分大小写

正则表达式匹配不区分大小写，为区分大小写，可使用 BINARY 关键字。

9.2 OR 匹配

OR 条件使用 |, 比如我在上面的例子搜索 ID 为 177、277、377、477 的记录，则有：

```
SELECT customer_id,first_name,last_name
FROM customer
WHERE customer_id REGEXP '177|277|377|477';
```

Listing 69: 含 OR 匹配的正则表达式

输出的结果为：

```
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         177 | SAMANTHA   | DUNCAN    |
|         277 | OLGA       | JIMENEZ   |
|         377 | HOWARD     | FORTNER   |
|         477 | DAN        | PAINE     |
+-------------+------------+-----------+
4 rows in set (0.00 sec)
```

Listing 70: 含 OR 匹配的正则表达式结果

9.3 匹配几个字符之一

格式是通过指定一组用 [和] 括起来的字符来完成，可以与上一个输出具有相同的效果：

```
SELECT customer_id,first_name,last_name
FROM customer
WHERE customer_id REGEXP '[1234]77';
```

Listing 71: 匹配几个字符的正则表达式

输出的结果为：

```
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         177 | SAMANTHA   | DUNCAN    |
|         277 | OLGA       | JIMENEZ   |
|         377 | HOWARD     | FORTNER   |
|         477 | DAN        | PAINE     |
+-------------+------------+-----------+
4 rows in set (0.00 sec)
```

Listing 72: 匹配几个字符的正则表达式结果

Tips 9.3.1 | 与 [ ] 的对比

[ ] 是另一种形式的 OR 语句。事实上，正则表达式 [123]Ton 为 [1|2|3]Ton 的缩写，也可以使用后者。

Tips 9.3.2 否定字符集

在集合的开始处放置一个 ^ 即可（[^123] 却匹配除这些字符外的任何东西）。

9.4 匹配范围

Tips 9.4.1 匹配范围

1. **数字集合**：[0-9]
2. **字母集合**：[a-z]

由此上面的例子可以写成:

```
SELECT customer_id,first_name,last_name
FROM customer
WHERE customer_id REGEXP '[1-5]77';
```

Listing 73: 匹配范围的正则表达式

输出的结果为:

```
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         177 | SAMANTHA   | DUNCAN    |
|         277 | OLGA       | JIMENEZ   |
|         377 | HOWARD     | FORTNER   |
|         477 | DAN        | PAINE     |
|         577 | CLIFTON    | MALCOLM   |
+-------------+------------+-----------+
5 rows in set (0.00 sec)
```

Listing 74: 匹配范围的正则表达式结果

9.5 匹配特殊字符

Defination 9.5.1 转义（escaping）

匹配特殊字符，必须用\\为前导来查找特殊字符，这种处理就是所谓的转义。

表 2: **空白元字符**

元字符	说明
\\f	换页
\\n	换行
\\r	回车
\\t	制表
\\v	纵向制表

我们可以检索包含.CROUSE 的邮箱:

```
SELECT customer_id,first_name,last_name,email
FROM customer
WHERE email REGEXP '\\.CAS';
```

Listing 75: 匹配特殊字符的正则表达式

输出的结果为:

```
+-------------+------------+-----------+-----------------------------------------+
| customer_id | first_name | last_name | email                                   |
+-------------+------------+-----------+-----------------------------------------+
|         207 | GERTRUDE   | CASTILLO  | GERTRUDE.CASTILLO@sakilacustomer.org    |
|         282 | JENNY      | CASTRO    | JENNY.CASTRO@sakilacustomer.org         |
|         423 | ALFRED     | CASILLAS  | ALFRED.CASILLAS@sakilacustomer.org      |
+-------------+------------+-----------+-----------------------------------------+
3 rows in set (0.00 sec)
```

Listing 76: 匹配特殊字符的正则表达式结果

Tips 9.5.1 匹配

为了匹配反斜杠（\）字符本身，需要使用\\\。

9.6 匹配字符类

Defination 9.6.1 字符类（character class）

为更方便工作预定义的字符集。

表 3: **字符类**

类	说明
[:alnum:]	任意字母和数字（同 [a-zA-Z0-9]）
[:alpha:]	任意字符（同 [a-zA-Z]）
[:blank:]	任意字符（同 [a-zA-Z]）
[:cntrl:]	ASCII 控制字符（ASCII 0 到 31 和 127）
[:digit:]	任意数字（同 [0-9]）
[:graph:]	与 [:print:] 相同，但不包括空格
[:lower:]	任意小写字母（同 [a-z]）
[:print:]	任意可打印字符
[:punct:]	既不在 [:alnum:] 又不在 [:cntrl:] 中的任意字符
[:space:]	包括空格在内的任意空白字符（同 [\\f \\n \\r \\t \\v]）
[:upper:]	任意大写字母（同 [A-Z]）
[:xdigit:]	任意十六进制数字（同 [a-fA-F0-9]）

9.7 匹配多个实例

表 4: **重复元字符**

元字符	说明
*	0 个或多个匹配
+	1 个或多个匹配（等于 1,）
?	0 个或 1 个匹配（等于 0,1）
{n}	指定数目的匹配
{n,}	不少于指定数目的匹配
{n,m}	匹配数目的范围（m 不超过 255）

例如，可以寻找 rating 为'PG','PG-13' 的记录：

```
SELECT FID,title,rating
FROM film_list
WHERE rating REGEXP 'PG*'
LIMIT 10;
```

Listing 77: 匹配多个实例的正则表达式

输出的结果为：

```
+------+------------------+--------+
| FID  | title            | rating |
+------+------------------+--------+
|    1 | ACADEMY DINOSAUR | PG     |
|    6 | AGENT TRUMAN     | PG     |
|    7 | AIRPLANE SIERRA  | PG-13  |
|    9 | ALABAMA DEVIL    | PG-13  |
|   12 | ALASKA PHANTOM   | PG     |
|   13 | ALI FOREVER      | PG     |
|   18 | ALTER VICTORY    | PG-13  |
|   19 | AMADEUS HOLY     | PG     |
|   28 | ANTHEM LUKE      | PG-13  |
|   33 | APOLLO TEEN      | PG-13  |
+------+------------------+--------+
```

Listing 78: 匹配多个实例的正则表达式结果

9.8 定位符

表 5: **定位元字符**

元字符	说明
^	文本的开始
$	文本的结尾
[[:<:]]	词的开始
[[:>:]]	词的结尾

我们尝试完全实现 LIKE 的功能：

```
SELECT first_name, last_name
FROM customer
WHERE first_name REGEXP '^ERI.$';
```

Listing 79: 匹配定位符的正则表达式

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ERIN       | DUNN      |
| ERIC       | ROBERT    |
| ERIK       | GUILLEN   |
+------------+-----------+
3 rows in set (0.00 sec)
```

Listing 80: 匹配定位符的正则表达式结果

10 创建计算字段

10.1 计算字段

Defination 10.1 字段（field）

基本上与列（column）的意思相同, 数据库列一般称为列, 字段通常用在计算字段的连接上。

Defination 10.1 计算字段

计算字段不存在于数据库表中。计算字段是运行时在 SELECT 语句内创建的。

10.2 拼接字段

Defination 10.2 拼接（concatenate）

将值联结到一起构成单个值，MySQL 的 SELECT 语句中用 Concat() 函数来拼接两个列。

可以采用第八章最后 customer 表的例子，我们将 first_name 字段和 laste_name 字段连接起来：

```
SELECT Concat(first_name,' ',last_name,'(',email,')')
FROM customer
WHERE first_name LIKE 'ERI_';
```

Listing 81: 拼接字段语句

输出的结果为：

```
+-----------------------------------------------+
| Concat(first_name,' ',last_name,'(',email,')') |
+-----------------------------------------------+
| ERIN DUNN(ERIN.DUNN@sakilacustomer.org)       |
| ERIC ROBERT(ERIC.ROBERT@sakilacustomer.org)   |
| ERIK GUILLEN(ERIK.GUILLEN@sakilacustomer.org) |
+-----------------------------------------------+
3 rows in set (0.00 sec)
```

Listing 82: 拼接字段语句的结果

在此基础上，可以运用 RTrim() 函数删除多余空格进而整理数据：

```
SELECT Concat(RTrim(first_name),' ',RTrim(last_name),'(',RTrim(email),')')
FROM customer
WHERE first_name LIKE 'ERI_'
ORDER BY first_name DESC;
```

Listing 83: 整理拼接字段语句

输出的结果为：

```
+-----------------------------------------------------------------+
| Concat(RTrim(first_name),' ',RTrim(last_name),'(',RTrim(email),')') |
+-----------------------------------------------------------------+
| ERIN DUNN(ERIN.DUNN@sakilacustomer.org)                         |
| ERIK GUILLEN(ERIK.GUILLEN@sakilacustomer.org)                   |
| ERIC ROBERT(ERIC.ROBERT@sakilacustomer.org)                     |
+-----------------------------------------------------------------+
3 rows in set (0.00 sec)
```

Listing 84: 整理拼接字段语句的结果

Tips 10.2

Trim 函数 MySQL 除了支持 RTrim()（正如刚才所见，它去掉串右边的空格），还支持 LTrim()（去掉串左边的空格）以及 Trim()（去掉串左右两边的空格）。

SELECT 的拼接语句生成的字段没有新列的名字，因此我们可以通过别名来实现。

Defination 10.2 别名（alias）

一个字段或值的替换名。别名用 AS 关键字赋予。(别名有时也称为导出列（derived column）)

那么可以把新生成的字段命名为 contact，则有：

```
SELECT Concat(RTrim(first_name),' ',RTrim(last_name),'(',RTrim(email),')')  AS contact
FROM customer
WHERE first_name LIKE 'ERI_'
ORDER BY first_name DESC;
```

Listing 85: 命名拼接字段语句

输出的结果为：

```
+-------------------------------------------+
| contact                                   |
+-------------------------------------------+
| ERIN DUNN(ERIN.DUNN@sakilacustomer.org)       |
| ERIK GUILLEN(ERIK.GUILLEN@sakilacustomer.org) |
| ERIC ROBERT(ERIC.ROBERT@sakilacustomer.org)   |
+-------------------------------------------+
3 rows in set (0.00 sec)
```

Listing 86: 命名拼接字段语句的结果

10.3 执行算术计算

我们可以检索 id 为 7 的公司年限和薪水的乘积:

```
SELECT film_id,
       rental_duration,
       replacement_cost,
       rental_duration*replacement_cost AS payment
FROM film
WHERE film_id=7;
```

Listing 87: 执行算术计算语句

输出的结果为:

```
+---------+-----------------+------------------+---------+
| film_id | rental_duration | replacement_cost | payment |
+---------+-----------------+------------------+---------+
|       7 |               6 |            28.99 |  173.94 |
+---------+-----------------+------------------+---------+
1 row in set (0.00 sec)
```

Listing 88: 执行算术计算语句的结果

表 6: **MySQL 算术操作符**

操作符	说明
+	加
-	减
*	乘
/	除

Tips 10.3.1 测试计算

1. SELECT 可以省略 FROM 子句以便简单地访问和处理表达式 (SELECT3*2; 将返回 6)。
2. SELECTNow() 利用 Now() 函数返回当前日期和时间。

11 数据处理函数

11.1 文本处理函数

可以运用文本处理函数返回字符串前 3 个字符:

```
SELECT film_id, title, LEFT(title, 3) AS sutitle
FROM film_text
WHERE film_id REGEXP '^[1-5]7$';
```

Listing 89: 执行文本处理函数语句

输出的结果为:

```
+---------+----------------+---------+
| film_id | title          | sutitle |
+---------+----------------+---------+
|      17 | ALONE TRIP     | ALO     |
|      27 | ANONYMOUS HUMAN | ANO     |
|      37 | ARIZONA BANG   | ARI     |
|      47 | BABY HALL      | BAB     |
|      57 | BASIC EASY     | BAS     |
+---------+----------------+---------+
5 rows in set (0.00 sec)
```

Listing 90: 执行文本处理函数语句的结果

表 7: **常用的文本处理函数**

函数	说明
Left()	返回串左边的字符
Length()	返回串的长度
Locate()	找出串的一个子串
Lower()	将串转换为小写
LTrim()	去掉串左边的空格
Right()	返回串右边的字符
RTrim()	去掉串右边的空格
Soundex()	返回串的 SOUNDEX 值
SubString()	返回子串的字符
Upper()	将串转换为大写

Tips 11.1.1 SOUNDEX

SOUNDEX 是一个将任何文本串转换为描述其语音表示的字母数字模式的算法。SOUNDEX 考虑了类似的发音字符和音节，使得能对串进行发音比较而不是字母比较。

11.2 日期和时间处理函数

DataBase 中，日期和时间采用相应的数据类型和特殊的格式存储。

Tips 11.2.1 日期格式

1. 日期必须为格式 yyyy-mm-dd。
2. 应该总是使用 4 位数字的年份，尽管 MYSQL 支持 2 位数字的年份，(MySQL 处理 00-69 为 2000-2069，处理 70-99 为 1970-1999)。
3. 如果你想要的仅是日期，则使用 Date()。

表 8: **常用日期和时间处理函数**

函数	说明
AddDate()	增加一个日期（天、周等）
AddTime()	增加一个时间（时、分等）
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算两个日期之差
Date_Add()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期，返回对应的星期几
Hour()	返回一个时间的小时部分
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期和时间
Second()	返回一个时间的秒部分
Time()	返回一个日期时间的时间部分
Year()	返回一个日期的年份部分

为了说明，现在可以检索 address 表中 30 分 0 秒至 30 分 3 秒生成的记录：

```
SELECT address,last_update
FROM address
WHERE Minute(last_update)=30
AND Second(last_update) BETWEEN 0 AND 3;
```

Listing 91: 执行日期和时间处理函数语句

输出的结果为：

```
+-----------------------------+---------------------+
| address                     | last_update         |
+-----------------------------+---------------------+
| 757 Rustenburg Avenue       | 2014-09-25 22:30:01 |
| 1892 Nabereznyje Telny Lane | 2014-09-25 22:30:02 |
| 1368 Maracabo Boulevard     | 2014-09-25 22:30:03 |
| 368 Hunuco Boulevard        | 2014-09-25 22:30:03 |
| 486 Ondo Parkway            | 2014-09-25 22:30:02 |
+-----------------------------+---------------------+
5 rows in set (0.00 sec)
```

Listing 92: 执行日期和时间处理函数语句的结果

11.3 数值处理函数

表 9: **常用数值处理函数**

函数	说明
Abs()	返回一个数的绝对值
Cos()	返回一个角度的余弦
Exp()	返回一个数的指数值
Mod()	返回除操作的余数
Pi()	返回圆周率
Rand()	返回一个随机数
Sin()	返回一个角度的正弦
Sqrt()	返回一个数的平方根
Tan()	返回一个角度的正切

12 汇总数据

12.1 聚集函数

Defination 12.1 聚集函数（aggregate function）

运行在行组上，计算和返回单个值的函数。

表 10: **SQL 聚集函数**

函数	说明
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()	返回某列值之和

12.2 AVG() 函数

看以下 SQL 语句：

```
SELECT AVG(amount) AS avg_amount FROM payment;
```

Listing 93: 执行 AVG 函数语句

输出的结果为：

```
+------------+
| avg_amount |
+------------+
|   4.201356 |
+------------+
1 row in set (0.00 sec)
```

Listing 94: 执行 AVG 函数语句的结果

Tips 12.2.1 AVG() 函数注意事项

1. AVG() 只能用来确定特定数值列的平均值，而且列名必须作为函数参数给出。
2. AVG() 函数忽略列值为 NULL 的行。

12.3 COUNT() 函数

看以下 SQL 语句：

```
SELECT COUNT(amount) AS count_amount FROM payment;
```

Listing 95: 执行 COUNT 函数语句

输出的结果为：

```
+--------------+
| count_amount |
+--------------+
|        16044 |
+--------------+
1 row in set (0.00 sec)
```

Listing 96: 执行 COUNT 函数语句的结果

Tips 12.3.1 COUNT() 函数注意事项

如果指定列名，则指定列的值为空的行被 COUNT() 函数忽略，但如果 COUNT() 函数中用的是星号（*），则不忽略。

12.4 MAX() 函数

看以下 SQL 语句：

```
SELECT MAX(amount) AS max_amount FROM payment;
```

Listing 97: 执行 MAX 函数语句

输出的结果为：

```
+------------+
| max_amount |
+------------+
|      11.99 |
+------------+
1 row in set (0.00 sec)
```

Listing 98: 执行 MAX 函数语句的结果

Tips 12.3.1 MAX() 函数注意事项

1. MySQL 允许将它用来返回任意列中的最大值，包括返回文本列中的最大值。

2. MAX() 函数忽略列值为 NULL 的行。

12.5 MIN() 函数

MIN() 函数与 MAX() 函数类似，具体可以看以下例子：

```
SELECT MIN(amount) AS min_amount FROM payment;
```

Listing 99: 执行 MIN 函数语句

输出的结果为：

```
+------------+
| min_amount |
+------------+
|       0.00 |
+------------+
1 row in set (0.00 sec)
```

Listing 100: 执行 MIN 函数语句的结果

12.6 SUM() 函数

```
SELECT SUM(amount) AS min_amount FROM payment WHERE payment_id REGEXP '^[1-7]7$';
```

Listing 101: 执行 SUM 函数语句

输出的结果为：

```
+------------+
| min_amount |
+------------+
|      31.93 |
+------------+
1 row in set (0.01 sec)
```

Listing 102: 执行 SUM 函数语句的结果

12.7 组合聚集函数

可以把之前的语句组合起来：

```
SELECT AVG(amount) AS avg_amount,
       COUNT(amount) AS count_amount,
       MAX(amount) AS max_amount,
       MIN(amount) AS min_amount
FROM payment;
```

Listing 103: 组合聚集函数语句

输出的结果为：

```
+------------+--------------+------------+------------+
| avg_amount | count_amount | max_amount | min_amount |
+------------+--------------+------------+------------+
|   4.201356 |        16044 |      11.99 |       0.00 |
+------------+--------------+------------+------------+
1 row in set (0.00 sec)
```

Listing 104: 组合聚集函数语句的结果

13 分组数据

13.1 创建分组

分组是在 SELECT 语句的 GROUP BY 子句中建立的，我们可以看两个 store 中的 inventory 数量：

```
SELECT store_id,COUNT(inventory_id) AS num_inevntory FROM inventory GROUP BY store_id;
```

Listing 105: 创建分组语句

输出的结果为：

```
+----------+---------------+
| store_id | num_inevntory |
+----------+---------------+
|        1 |          2270 |
|        2 |          2311 |
+----------+---------------+
2 rows in set (0.00 sec)
```

Listing 106: 创建分组语句的结果

Tips 13.1.1 GROUP BY 子句注意事项

1. GROUP BY 子句可以包含任意数目的列，这使得能对分组进行嵌套。
2. 在 GROUP BY 子句中嵌套了分组，数据将在最后规定的分组上进行汇总。
3. GROUP BY 子句中列出的每个列都必须是检索列或有效的表达式（但不能是聚集函数）。
4. 如果在 SELECT 中使用表达式，则必须在 GROUP BY 子句中指定相同的表达式。不能使用别名。
5. 除聚集计算语句外，SELECT 语句中的每个列都必须在 GROUP BY 子句中给出。
6. 分组列中具有 NULL 值，则 NULL 将作为一个分组返回。如果列中有多行 NULL 值，它们将分为一组。
7. GROUP BY 子句在 WHERE 子句之后，ORDER BY 子句之前。

使用 WITH ROLLUP 可以得到分组汇总的值：

```
SELECT store_id,COUNT(inventory_id) AS num_inevntory
FROM inventory
GROUP BY store_id
WITH ROLLUP;
```

Listing 107: 使用 WITH ROLLUP 语句

输出的结果为：

```
+----------+---------------+
| store_id | num_inevntory |
+----------+---------------+
|        1 |          2270 |
|        2 |          2311 |
|     NULL |          4581 |
+----------+---------------+
3 rows in set (0.00 sec)
```

Listing 108: 使用 WITH ROLLUP 语句的结果

13.2 过滤分组

Tips 13.2.1 HAVING 注意事项

1. WHERE 过滤指定的是行而不是分组。
2. HAVING 支持所有 WHERE 操作符。
3. WHERE 在数据分组前进行过滤，HAVING 在数据分组后进行过滤。

可以看到以下的 SQL 语句：

```
SELECT film_id,COUNT(inventory_id) AS num_inevntory
FROM inventory GROUP BY film_id
HAVING COUNT(inventory_id) = 7
And film_id REGEXP '^[1-9]9.$';
```

Listing 109: 使用过滤分组语句

输出的结果为：

```
+---------+---------------+
| film_id | num_inevntory |
+---------+---------------+
|     391 |             7 |
|     397 |             7 |
|     590 |             7 |
|     698 |             7 |
|     790 |             7 |
|     890 |             7 |
|     891 |             7 |
|     892 |             7 |
|     895 |             7 |
|     993 |             7 |
+---------+---------------+
10 rows in set (0.00 sec)
```

Listing 110: 使用过滤分组语句的结果

13.3 分组和排序

表 11: **ORDER BY 与 GROUP BY 的对比**

ORDER BY	GROUP BY
排序产生的输出	分组行，但输出可能不是分组的顺序
任意列都可以使用（甚至非选择的列也可以使用）	只可能使用选择列或表达式列，而且必须使用每个选择列表达式
不一定需要	如果与聚集函数一起使用（或表达式），则必须使用

现在将上面那个例子用降序输出：

```
SELECT film_id,COUNT(inventory_id) AS num_inevntory
FROM inventory GROUP BY film_id
HAVING COUNT(inventory_id) = 7
And film_id REGEXP '^[1-9]9.$'
ORDER BY film_id DESC;
```

Listing 111: 使用过滤分组降序语句

输出的结果为：

```
+---------+---------------+
| film_id | num_inevntory |
+---------+---------------+
|     993 |             7 |
|     895 |             7 |
|     892 |             7 |
|     891 |             7 |
|     890 |             7 |
|     790 |             7 |
|     698 |             7 |
|     590 |             7 |
|     397 |             7 |
|     391 |             7 |
+---------+---------------+
10 rows in set (0.00 sec)
```

Listing 112: 使用过滤分组语句的结果

13.4 SELECT 子句顺序

表 12: **SQL 聚集函数**

子句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否
LIMIT	要检索的行数	否

14 使用子查询

14.1 利用子查询进行过滤

Defination 14.1.1 查询（query）

任何 SQL 语句都是查询。但此术语一般指 SELECT 语句。

Defination 14.1.2 子查询（subquery）

嵌套在其他查询中的查询。

例如，我想知道顾客 id 为 17、27、37、47、57、67、77、87、97 的地址，但是 customer 表中只有地址的 id，那么我们就可以采用子查询：

```
SELECT address
FROM address
WHERE address_id IN ( SELECT address_id
                      FROM customer
                      WHERE customer_id REGEXP '^[1-9]7$');
```

Listing 113: 使用子查询的语句

输出的结果为：

```
+-----------------------+
| address               |
+-----------------------+
| 270 Toulon Boulevard  |
| 217 Botshabelo Place  |
| 1440 Fukuyama Loop    |
| 686 Garland Manor     |
| 943 Tokat Street      |
| 1586 Guaruj Place     |
| 1692 Ede Loop         |
| 1740 Portoviejo Avenue |
| 1599 Plock Drive      |
+-----------------------+
9 rows in set (0.00 sec)
```

Listing 114: 使用子查询的语句的结果

Tips 14.1.1 子查询注意事项

1. 在 WHERE 子句中使用子查询，该保证 SELECT 语句具有与 WHERE 子句中相同数目的列。
2. 虽然子查询一般与 IN 操作符结合使用，但也可以用于测试等于（=）、不等于（<>）等。
3. 子查询不算一种特别有效的方法，尽量不要用。

14.2 作为计算字段使用子查询

例如，我们可以执行把每个 staff 经手的订单加起来：

```
SELECT staff_id,first_name,last_name,(SELECT COUNT(amount)
                                      FROM payment
                                      WHERE payment.staff_id = staff.staff_id ) AS sum_payment
FROM staff
ORDER BY staff_id;
```

Listing 115: 作为计算字段使用子查询的语句

输出的结果为：

```
+----------+------------+-----------+-------------+
| staff_id | first_name | last_name | sum_payment |
+----------+------------+-----------+-------------+
|        1 | Mike       | Hillyer   |        8054 |
|        2 | Jon        | Stephens  |        7990 |
+----------+------------+-----------+-------------+
2 rows in set (0.01 sec)
```

Listing 116: 作为计算字段使用子查询的语句的结果

Defination 14.2.1 相关子查询（correlated subquery）

涉及外部查询的子查询。

其中子查询的 WHERE 语句就是相关子查询，任何时候只要列名可能有多义性，就必须使用这种语法（表名和列名由一个句点分隔）。如果不采用这种完全限定列名，那么 MySQL 就会把 payment 表中 staff_id 与自身比较，从而出错：

```
SELECT staff_id,first_name,last_name,(SELECT COUNT(amount)
                                      FROM payment
                                      WHERE staff_id = staff_id ) AS sum_payment
FROM staff
ORDER BY staff_id;
```

Listing 117: 作为计算字段错误使用子查询的语句

输出的结果为：

```
+----------+------------+-----------+-------------+
| staff_id | first_name | last_name | sum_payment |
+----------+------------+-----------+-------------+
|        1 | Mike       | Hillyer   |       16044 |
|        2 | Jon        | Stephens  |       16044 |
+----------+------------+-----------+-------------+
2 rows in set (0.01 sec)
```

Listing 118: 作为计算字段错误使用子查询的语句的结果

15 联结表

15.1 联结

Defination 15.1.1 外键（foreign key）

外键为某个表中的一列，它包含另一个表的主键值，定义了两个表之间的关系。

Defination 15.1.2 可伸缩性（scale）

能够适应不断增加的工作量而不失败。设计良好的数据库或应用程序称之为可伸缩性好（scale well）。

15.2 等值联结（equijoin）

例如，可以联结 address 表和 city 表，输出前十个 address 和 city 的名称：

```
SELECT address,city,district
FROM address,city
WHERE address.city_id = city.city_id
ORDER BY address,city
LIMIT 10;
```

Listing 119: 创建联结的语句

输出的结果为：

```
+--------------------------------+-------------+--------------+
| address                        | city        | district     |
+--------------------------------+-------------+--------------+
| 1 Valle de Santiago Avenue     | Brindisi    | Apulia       |
| 1001 Miyakonojo Lane           | Taizz       | Taizz        |
| 1002 Ahmadnagar Manor          | Huixquilucan| Mxico        |
| 1003 Qinhuangdao Street        | Purwakarta  | West Java    |
| 1006 Santa Brbara dOeste Manor | Owo         | Ondo & Ekiti |
| 1009 Zanzibar Lane             | Arecibo     | Arecibo      |
| 1010 Klerksdorp Way            | Graz        | Steiermark   |
| 1013 Tabuk Boulevard           | Kanchrapara | West Bengali |
| 1014 Loja Manor                | Ambattur    | Tamil Nadu   |
| 1016 Iwakuni Street            | Kingstown   | St George    |
+--------------------------------+-------------+--------------+
10 rows in set (0.00 sec)
```

Listing 120: 创建联结的语句的结果

Tips 15.2.1 、注意完全限定列名

1. 在引用的列可能出现二义性时，必须使用完全限定列名（用一个点分隔的表名和列名）。
2. 在联结两个表时，你实际上做的是将第一个表中的每一行与第二个表中的每一行配对。WHERE 子句作为过滤条件，它只包含那些匹配给定条件（这里是联结条件）的行。
3. 应该保证所有联结都有 WHERE 子句，否则 MySQL 将返回比想要的数据多得多的数据。
4. 有时我们会听到返回称为叉联结（cross join）的笛卡儿积的联结类型。

Defination 15.2.1 笛卡儿积（cartesian product）

由没有联结条件的表关系返回的结果为笛卡儿积。检索出的行的数目将是第一个表中的行数乘以第二个表中的行数。

下面用 staff 表和 address 表来演示以下没有完全限定列名和有限定列名的情况：

```
SELECT first_name,last_name,address
FROM staff,address
ORDER BY address,first_name
LIMIT 30;
```

Listing 121: 没有使用完全限定列名的创建联结的语句

输出的结果为：

```
+------------+-----------+---------------------------------+
| first_name | last_name | address                         |
+------------+-----------+---------------------------------+
| Jon        | Stephens  | 1 Valle de Santiago Avenue      |
| Mike       | Hillyer   | 1 Valle de Santiago Avenue      |
| Jon        | Stephens  | 1001 Miyakonojo Lane            |
| Mike       | Hillyer   | 1001 Miyakonojo Lane            |
| Jon        | Stephens  | 1002 Ahmadnagar Manor           |
| Mike       | Hillyer   | 1002 Ahmadnagar Manor           |
| Jon        | Stephens  | 1003 Qinhuangdao Street         |
| Mike       | Hillyer   | 1003 Qinhuangdao Street         |
| Jon        | Stephens  | 1006 Santa Brbara dOeste Manor  |
| Mike       | Hillyer   | 1006 Santa Brbara dOeste Manor  |
| Jon        | Stephens  | 1009 Zanzibar Lane              |
| Mike       | Hillyer   | 1009 Zanzibar Lane              |
| Jon        | Stephens  | 1010 Klerksdorp Way             |
| Mike       | Hillyer   | 1010 Klerksdorp Way             |
| Jon        | Stephens  | 1013 Tabuk Boulevard            |
| Mike       | Hillyer   | 1013 Tabuk Boulevard            |
| Jon        | Stephens  | 1014 Loja Manor                 |
| Mike       | Hillyer   | 1014 Loja Manor                 |
| Jon        | Stephens  | 1016 Iwakuni Street             |
| Mike       | Hillyer   | 1016 Iwakuni Street             |
| Jon        | Stephens  | 102 Chapra Drive                |
| Mike       | Hillyer   | 102 Chapra Drive                |
| Jon        | Stephens  | 1027 Banjul Place               |
| Mike       | Hillyer   | 1027 Banjul Place               |
| Jon        | Stephens  | 1027 Songkhla Manor             |
| Mike       | Hillyer   | 1027 Songkhla Manor             |
| Jon        | Stephens  | 1029 Dzerzinsk Manor            |
| Mike       | Hillyer   | 1029 Dzerzinsk Manor            |
| Jon        | Stephens  | 1031 Daugavpils Parkway         |
| Mike       | Hillyer   | 1031 Daugavpils Parkway         |
+------------+-----------+---------------------------------+
30 rows in set (0.00 sec)
```

Listing 122: 没有使用完全限定列名的创建联结的语句的结果

最终实际生成了 1206 条结果，显然 staff 有两条记录，我们可以查询一下 address 的行数：

```
SELECT COUNT(address) AS num_address FROM address;
```

Listing 123: 查询 staff 行数的语句

输出的结果为：

```
+-------------+
| num_address |
+-------------+
|         603 |
+-------------+
1 row in set (0.00 sec)
```

Listing 124: 查询 staff 行数的语句的结果

显然，staff 有 603 条，那么所谓笛卡尔积就是 2*603=1206 条，显然是不对的，下面展示正确的 SQL 语句：

```
SELECT first_name,last_name,address
FROM staff,address
WHERE staff.address_id = address.address_id
ORDER BY first_name,last_name;
```

Listing 125: 使用完全限定列名的创建联结的语句

输出的结果为：

```
+------------+-----------+----------------------+
| first_name | last_name | address              |
+------------+-----------+----------------------+
| Jon        | Stephens  | 1411 Lillydale Drive |
| Mike       | Hillyer   | 23 Workhaven Lane    |
+------------+-----------+----------------------+
2 rows in set (0.00 sec)
```

Listing 126: 使用完全限定列名的语句的结果

15.3 内部联结

等值联结（equijoin）是基于两个表之间的相等测试。这种联结也称为内部联结。因此我们可以用下面的语句返回和上面例子同样的结果：

```
SELECT first_name,last_name,address
FROM staff INNER JOIN address
ON staff.address_id = address.address_id
ORDER BY first_name,last_name;
```

Listing 127: 使用内部联结的语句

输出的结果为：

```
+------------+-----------+---------------------+
| first_name | last_name | address             |
+------------+-----------+---------------------+
| Jon        | Stephens  | 1411 Lillydale Drive |
| Mike       | Hillyer   | 23 Workhaven Lane    |
+------------+-----------+---------------------+
2 rows in set (0.00 sec)
```

Listing 128: 使用内部联结的语句的结果

15.4 联结多个表

```
SELECT title, name
FROM film_category, film, category
WHERE film_category.film_id = film.film_id
AND film_category.category_id = category.category_id
AND film_category.film_id REGEXP '^[1-9]7$'
ORDER BY title, name;
```

Listing 129: 联结多个表的语句

输出的结果为:

```
+-------------------+----------+
| title             | name     |
+-------------------+----------+
| ALONE TRIP        | Music    |
| ANONYMOUS HUMAN   | Sports   |
| ARIZONA BANG      | Classics |
| BABY HALL         | Foreign  |
| BASIC EASY        | Travel   |
| BERETS AGENT      | Action   |
| BIRDS PERDITION   | New      |
| BOONDOCK BALLROOM | Travel   |
| BRIDE INTRIGUE    | Action   |
+-------------------+----------+
9 rows in set (0.00 sec)
```

Listing 130: 使用内部联结的语句的结果

16 创建高级联结

16.1 使用表别名

仍然采用上面的例子进行输出:

```
SELECT title, name
FROM film_category AS a, film AS b, category AS c
WHERE a.film_id = b.film_id
AND a.category_id = c.category_id
AND a.film_id REGEXP '^[1-9]7$'
ORDER BY title, name;
```

Listing 131: 使用表别名的语句

输出的结果为:

```
+-------------------+----------+
| title             | name     |
+-------------------+----------+
| ALONE TRIP        | Music    |
| ANONYMOUS HUMAN   | Sports   |
| ARIZONA BANG      | Classics |
| BABY HALL         | Foreign  |
| BASIC EASY        | Travel   |
| BERETS AGENT      | Action   |
| BIRDS PERDITION   | New      |
| BOONDOCK BALLROOM | Travel   |
| BRIDE INTRIGUE    | Action   |
+-------------------+----------+
9 rows in set (0.00 sec)
```

Listing 132: 使用表别名的语句的结果

Tips 16.1.1 注意表别名

表别名只在查询执行中使用。与列别名不一样，表别名不返回到客户机。

16.2 自联结

可以看以下例子，假如 id 为 577 的 customer 反映有买的东西有问题，现在需要找到这个顾客买的 inventory, 看看其他买这些东西的顾客是否有问题:

```
SELECT DISTINCT r1.customer_id
FROM rental AS r1,rental AS r2
WHERE r1.inventory_id = r2.inventory_id
AND r2.customer_id = 577
ORDER BY r1.customer_id
LIMIT 10;
```

Listing 133: 使用自联结的语句

输出的结果为:

```
+-------------+
| customer_id |
+-------------+
|          11 |
|          14 |
|          23 |
|          26 |
|          34 |
|          37 |
|          44 |
|          46 |
|          50 |
|          51 |
+-------------+
10 rows in set (0.00 sec)
```

Listing 134: 使用自联结的语句的结果

Tips 16.2.1 用自联结而不用子查询

自联结虽然最终的结果与子查询语句是相同的，但有时候处理联结远比处理子查询快得多。

16.3 自然联结

对内部联结的例子重新用自然联结写：

```
SELECT s.first_name,s.last_name,a.address
FROM staff AS s, address AS a
WHERE s.address_id = a.address_id
ORDER BY s.first_name,s.last_name;
```

Listing 135: 使用自然联结的语句

输出的结果为：

```
+------------+-----------+----------------------+
| first_name | last_name | address              |
+------------+-----------+----------------------+
| Jon        | Stephens  | 1411 Lillydale Drive |
| Mike       | Hillyer   | 23 Workhaven Lane    |
+------------+-----------+----------------------+
2 rows in set (0.00 sec)
```

Listing 136: 使用自然联结的语句的结果

自然联结只能选择那些唯一的列。这一般是通过对表使用通配符（SELECT *），对所有其他表的列使用明确的子集来完成的，迄今为止我们建立的每个内部联结都是自然联结，

16.4 外部联结

Defination 16.3.1 外部联结

联结包含了那些在相关表中没有关联行的行。

外部联结主要是 OUTER LEFT/RIGHT JOIN，用哪边就是输出所有行的表，例如，所有的电影都是英文的，现在我要检索出所有语言，就需要运用外联结：

```
SELECT l.name,f.title
FROM language AS l LEFT OUTER JOIN film AS f
ON l.language_id = f.language_id
ORDER BY l.language_id DESC
LIMIT 10;
```

Listing 137: 使用外联结的语句

输出的结果为：

```
+----------+-------------------+
| name     | title             |
+----------+-------------------+
| German   | NULL              |
| French   | NULL              |
| Mandarin | NULL              |
| Japanese | NULL              |
| Italian  | NULL              |
| English  | ZOOLANDER FICTION |
| English  | ZHIVAGO CORE      |
| English  | YOUNG LANGUAGE    |
| English  | YENTL IDAHO       |
| English  | WRONG BEHAVIOR    |
+----------+-------------------+
10 rows in set (0.00 sec)
```

Listing 138: 使用外联结的语句的结果

16.5 使用带聚集函数的联结

```
SELECT l.name,COUNT(f.film_id) AS num_film
FROM language AS l LEFT OUTER  JOIN film AS f
ON l.language_id = f.language_id
GROUP BY l.language_id
ORDER BY l.language_id;
```

Listing 139: 使用带聚集函数的语句

输出的结果为：

```
+----------+----------+
| name     | num_film |
+----------+----------+
| English  |     1000 |
| Italian  |        0 |
| Japanese |        0 |
| Mandarin |        0 |
| French   |        0 |
| German   |        0 |
+----------+----------+
6 rows in set (0.00 sec)
```

Listing 140: 使用带聚集函数的语句的结果

Tips 16.5.1 外联结注意事项

1. 没有 *= 操作符，MySQL 不支持简化字符 *= 和 =* 的使用。
2. 左右联结可以相互转换。

17 组合查询

17.1 组合查询

Defination 17.1.1 复合查询（compound query）或并（union）

执行多个查询（多条 SELECT 语句），并将结果作为单个查询结果集返回。

Tips 17.1.2 组合查询注意事项

组合相同表的两个查询完成的工作与具有多个 WHERE 子句条件的单条查询完成的工作相同。

17.2 使用 UNION

可以看以下例子，分别对 address_id 和 customer_id 施加条件：

```
SELECT first_name,last_name
FROM customer
WHERE address_id REGEXP '^[1-5]7$'
UNION SELECT first_name,last_name
FROM customer
WHERE customer_id LIKE '7_'
ORDER BY first_name;
```

Listing 141: 使用 UNION 的语句

输出的结果为：

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ANNA       | HILL      |
| BEVERLY    | BROOKS    |
| CHRISTINA  | RAMIREZ   |
| CHRISTINE  | ROBERTS   |
| DENISE     | KELLY     |
| HEATHER    | MORRIS    |
| IRENE      | PRICE     |
| JANE       | BENNETT   |
| KAREN      | JACKSON   |
| KATHY      | JAMES     |
| LORI       | WOOD      |
| RACHEL     | BARNES    |
| SARAH      | LEWIS     |
| TAMMY      | SANDERS   |
| THERESA    | WATSON    |
+------------+-----------+
15 rows in set (0.00 sec)
```

Listing 142: 使用 UNION 的语句的结果

上面的结果可以用多条 WHERE 子句完成:

```
SELECT first_name,last_name
FROM customer
WHERE address_id REGEXP '^[1-5]7$'
OR customer_id LIKE '7_'
ORDER BY first_name;
```

Listing 143: 使用 WHERE 复现的语句

输出的结果为:

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ANNA       | HILL      |
| BEVERLY    | BROOKS    |
| CHRISTINA  | RAMIREZ   |
| CHRISTINE  | ROBERTS   |
| DENISE     | KELLY     |
| HEATHER    | MORRIS    |
| IRENE      | PRICE     |
| JANE       | BENNETT   |
| KAREN      | JACKSON   |
| KATHY      | JAMES     |
| LORI       | WOOD      |
| RACHEL     | BARNES    |
| SARAH      | LEWIS     |
| TAMMY      | SANDERS   |
| THERESA    | WATSON    |
+------------+-----------+
15 rows in set (0.00 sec)
```

Listing 144: 使用 WHERE 复现的语句的结果

Tips 17.2.1 UNION 规则

1. UNION 必须由两条或两条以上的 SELECT 语句组成，语句之间用关键字 UNION 分隔。
2. UNION 中的每个查询必须包含相同的列、表达式或聚集函数（不过各个列不需要以相同的次序列出）。
3. 列数据类型必须兼容：类型不必完全相同，但必须是 DBMS 可以隐含地转换的类型。

17.3 包含或取消重复的行

Tips 17.3.1 UNION 包含或取消重复的行

1. UNION 从查询结果集中自动去除了重复的行。
2. 如果想返回所有匹配行，可使用 UNION ALL 而不是 UNION。

我将上面例子改一下，用下面的语句反应出有重复行和没有重复行的区别：

```
SELECT first_name,last_name
FROM customer
WHERE address_id REGEXP '^[1-5]7$'
UNION SELECT first_name,last_name
FROM customer
WHERE customer_id LIKE '_3'
ORDER BY first_name;
```

Listing 145: 使用 UNION 不包含重复行的语句

输出的结果为：

```
+------------+------------+
| first_name | last_name  |
+------------+------------+
| ANNA       | HILL       |
| ASHLEY     | RICHARDSON |
| BEVERLY    | BROOKS     |
| CHRISTINE  | ROBERTS    |
| HEATHER    | MORRIS     |
| KAREN      | JACKSON    |
| LOUISE     | JENKINS    |
| PHYLLIS    | FOSTER     |
| SARAH      | LEWIS      |
+------------+------------+
9 rows in set (0.00 sec)
```

Listing 146: 使用 UNION 不包含重复行的语句的结果

下面是包含重复行的:

```
SELECT first_name,last_name
FROM customer
WHERE address_id REGEXP '^[1-5]7$'
UNION ALL SELECT first_name,last_name
FROM customer
WHERE customer_id LIKE '_3'
ORDER BY first_name;
```

Listing 147: 使用 UNION 包含重复行的语句

输出的结果为:

```
+------------+------------+
| first_name | last_name  |
+------------+------------+
| ANNA       | HILL       |
| ANNA       | HILL       |
| ASHLEY     | RICHARDSON |
| BEVERLY    | BROOKS     |
| CHRISTINE  | ROBERTS    |
| CHRISTINE  | ROBERTS    |
| HEATHER    | MORRIS     |
| HEATHER    | MORRIS     |
| KAREN      | JACKSON    |
| KAREN      | JACKSON    |
| LOUISE     | JENKINS    |
| PHYLLIS    | FOSTER     |
| SARAH      | LEWIS      |
| SARAH      | LEWIS      |
+------------+------------+
14 rows in set (0.00 sec)
```

Listing 148: 使用 UNION 包含重复行的语句的结果

显然多了 5 条记录，这就是 UNION ALL 的作用。

17.4 对组合查询结果排序

Tips 17.4.1 组合查询结果排序注意事项

在用 UNION 组合查询时，只能使用一条 ORDER BY 子句，它必须出现在最后一条 SELECT 语句之后。

本章所有的语句的例子都使用了 ORDER BY，这里就不赘述举例了。

Tips 17.4.2 组合不同的表

使用 UNION 的组合查询可以应用不同的表，而不仅仅用于相同的表。

18 全文本搜索

18.1 启用全文本搜索支持

一般在创建表时启用全文本搜索. 如下所示：

```
CREATE TABLE productnotes
(
        note_id int                             NOT NULL AUTO INCREMENT,
        prod_id char(10)                        NOT NULL,
        note_date datetime                      NOT NULL,
        note_text text                          NULL,
        PRIMARY KEY(note_id),
        FULLTEXT(note_text)
)ENGINE=MYISAM:
```

Listing 149: 启用全文本搜索支持的语句

Tips 18.1.1 FULLTEXT 注意事项

1. 可以在创建表时指定 FULLTEXT，或者在稍后指定（在这种情况下所有已有数据必须立即索引）。
2. 不要在导入数据时使用 FULLTEXT。

18.2 进行全文本搜索

其中 Match() 指定被搜索的列，Against() 指定要使用的搜索表达式。

```
SELECT description
FROM film
WHERE MATCH(description) AGAINST('Monastery')
ORDER BY description
LIMIT 10;
```

Listing 150: 进行全文本搜索的语句

输出的结果为：

```
+-----------------------------------------------------------------------------------------------------------------+
| description                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------+
| A Action-Packed Yarn of a Womanizer And a Lumberjack who must Chase a Sumo Wrestler in A Monastery              |
| A Amazing Story of a Mad Cow And a Dog who must Kill a Husband in A Monastery                                   |
| A Astounding Character Study of a Composer And a Student who must Overcome a Composer in A Monastery           |
| A Astounding Character Study of a Womanizer And a Hunter who must Escape a Robot in A Monastery                 |
| A Astounding Documentary of a Butler And a Explorer who must Challenge a Butler in A Monastery                  |
| A Awe-Inspiring Story of a Feminist And a Cat who must Conquer a Dog in A Monastery                              |
| A Beautiful Character Study of a Robot And a Astronaut who must Overcome a Boat in A Monastery                  |
| A Beautiful Documentary of a Astronaut And a Crocodile who must Discover a Madman in A Monastery                |
| A Brilliant Panorama of a Mad Scientist And a Mad Cow who must Meet a Pioneer in A Monastery                    |
| A Fanciful Story of a Man And a Sumo Wrestler who must Outrace a Student in A Monastery                         |
+-----------------------------------------------------------------------------------------------------------------+
10 rows in set (0.00 sec)
```

Listing 151: 进行全文本搜索的语句的结果

Tips 18.2.1 MATCH 注意事项

1. 使用完整的 Match(), 说明传递给 Match() 的值必须与 FULLTEXT() 定义中的相同。
2. 除非使用 BINARY 方式（本章中没有介绍），否则全文本搜索不区分大小写。
3. （使用 LIKE）以不特别有用的顺序返回数据。前者（使用全文本搜索）返回以文本匹配的良好程度排序的数据。

```
SELECT description,
       MATCH(description) AGAINST('Monastery') AS relevance_score
FROM film
LIMIT 5;
```

Listing 152: 进行全文本搜索排序的语句

输出的结果为 (太长处理了一下，但大概意思是这样)：

```
+----------------------------------------+-----------------+
| description                            | relevance_score |
+----------------------------------------+-----------------+
| A Epic Drama of a Feminist And a Mad Scie|               0 |
| A Astounding Epistle of a Database Admini|               0 |
| A Astounding Reflection of a Lumberjack A|               0 |
|a Boy who must Redeem a Boy in A Monastery| 2.15655422210693|
+----------------------------------------+-----------------+
5 rows in set (0.00 sec)
```

Listing 153: 进行全文本搜索排序的语句的结果

Tips 18.2.2 计算出来的等级值

1. 不包含词的行等级为 0（因此不被前一例子中的 WHERE 子句选择）。
2. 包含词的每行都有一个等级值，文本中词靠前的行的等级值比词靠后的行的等级值高。

18.3 使用查询扩展

采用之前的第一个例子：

```
SELECT description
FROM film
WHERE MATCH(description) AGAINST('Monastery' WITH QUERY EXPANSION)
LIMIT 10;
```

Listing 154: 使用查询扩展的语句

输出的结果为：

```
+--------------------------------------------------------------------------+
| description                                                              |
+--------------------------------------------------------------------------+
| A Action-Packed Yarn of a Womanizer And a Lumberjack who must Chase a    |
|Sumo Wrestler in A Monastery                                              |
| A Unbelieveable Character Study of a Cat And a Database Administrator    |
|who must Pursue a Teacher in A Monastery                                  |
| A Intrepid Reflection of a Technical Writer And a Hunter who must        |
|Defeat a Sumo Wrestler in A Monastery                                     |
| A Action-Packed Story of a Pioneer And a Technical Writer who must       |
|Discover a Forensic Psychologist in An Abandoned Amusement Park           |
| A Awe-Inspiring Drama of a Technical Writer And a Composer who must      |
|Reach a Pastry Chef in A U-Boat                                           |
| A Stunning Character Study of a Mad Scientist And a Mad Cow who must     |
|Kill a Car in A Monastery                                                 |
| A Awe-Inspiring Character Study of a Robot And a Sumo Wrestler who       |
|must Discover a Womanizer in A Shark Tank                                 |
| A Awe-Inspiring Tale of a Forensic Psychologist And a Woman who must     |
|Challenge a Database Administrator in Ancient Japan                       |
| A Awe-Inspiring Reflection of a Pastry Chef And a Teacher who must       |
|Overcome a Sumo Wrestler in A U-Boat                                      |
| A Awe-Inspiring Documentary of a Robot And a Mad Scientist who must      |
|Reach a Database Administrator in A Shark Tank                            |
+--------------------------------------------------------------------------+
10 rows in set (0.01 sec)
```

Listing 155: 使用查询扩展的语句的结果

18.4 布尔文本搜索

Tips 18.4.1 布尔文本搜索可以提供的细节

1. 要匹配的词。
2. 要排斥的词（如果某行包含这个词，则不返回该行，即使它包含其他指定的词也是如此）。
3. 排列提示（指定某些词比其他词更重要，更重要的词等级更高）。
4. 表达式分组。
5. 另外一些内容。

Tips 18.4.2 没有 FULLTEXT 索引也可以使用

布尔方式不同于迄今为止使用的全文本搜索语法的地方在于，即使没有定义 FULLTEXT 索引，也可以使用它。

```
SELECT title FROM film WHERE MATCH(title) AGAINST('VIRGIN' IN BOOLEAN MODE);
```

Listing 156: 使用布尔文本搜索的语句

输出的结果为：

```
+--------------+
| title        |
+--------------+
| SEA VIRGIN   |
| VIRGIN DAISY |
+--------------+
2 rows in set (0.01 sec)
```

Listing 157: 使用布尔文本搜索的语句的结果

表 13: **全文本布尔操作符**

布尔操作符	说明
+	包含，词必须存在
-	排除，词必须不出现
>	包含，而且增加等级值
<	包含，且减少等级值
()	把词组成子表达式（允许这些子表达式作为一个组被包含、排除、排列等）
	取消一个词的排序值
*	词尾的通配符
""	定义一个短语（与单个词的列表不一样，它匹配整个短语以便包含或排除这个短语）

下面举个例子，我把上面那个例子要求不检索 DAISY：

```
SELECT title FROM film WHERE MATCH(title) AGAINST('VIRGIN -DAISY' IN BOOLEAN MODE);
```

Listing 158: 使用复杂布尔文本搜索的语句

输出的结果为：

```
+------------+
| title      |
+------------+
| SEA VIRGIN |
+------------+
1 row in set (0.01 sec)
```

Listing 159: 使用复杂布尔文本搜索的语句的结果

Tips 18.4.3 排列而不排序

布尔方式中，不按等级值降序排序返回的行。

Tips 18.4.4 全文本搜索的使用说明

1. 在索引全文本数据时，短词被忽略且从索引中排除。短词定义为那些具有 3 个或 3 个以下字符的词（如果需要，这个数目可以更改）。
2. MySQL 带有一个内建的非用词（stopword）列表，这些词在索引全文本数据时总是被忽略。

3. MySQL 规定了一条 50% 规则，如果一个词出现在 50% 以上的行中，则将它作为一个非用词忽略。
4. 如果表中的行数少于 3 行，则全文本搜索不返回结果（因为每个词或者不出现，或者至少出现在 50% 的行中）。
5. 忽略词中的单引号。
6. 不具有词分隔符（包括日语和汉语）的语言不能恰当地返回全文本搜索结果。

19 插入数据

19.1 插入完整的行

例如：

```
INSERT INTO customer
VALUES (NULL, 1, 'RED', 'BAMBOO', 'chenqw0409@163.com', 77, 1, NOW(), NOW());
```

Listing 160: 插入完整的行的语句

这个语句可以换用一种更加安全的方式：

```
INSERT INTO customer (customer_id,
                      store_id,
                      first_name,
                      last_name,
                      email,
                      address_id,
                      active,
                      create_date,
                      last_update)
VALUES (NULL,
        2,
        'CHEN',
        'QIWEI',
        'chenqw0621@edu.jlu.com',
        177,
        1,
        NOW(),
        NOW());
```

Listing 161: 更安全的插入完整的行的语句

Tips 19.1.1 插入列的注意事项

1. 总是使用列的列表，使用列的列表能使 SQL 代码继续发挥作用，即使表结构发生了变化。
2. 不管使用哪种 INSERT 语法，都必须给出 VALUES 的正确数目。
3. 果表的定义允许，则可以在 INSERT 操作中省略某些列。(字段运行空值或表定义给出默认值)
4. 如果数据检索是最重要的，则可以通过在 INSERT 和 INTO 之间添加关键字 LOW_PRIORITY，指示

MySQL 降低 INSERT 语句的优先级。

19.2 插入多个行

可以直接采用以下例子:

```
INSERT INTO customer (customer_id,
                      store_id,
                      first_name,
                      last_name,
                      email,
                      address_id,
                      active,
                      create_date,
                      last_update)
VALUES (NULL,
        2,
        'BLUE',
        'BAMBOO',
        'chenqw0621@edu.jlu.com',
        277,
        2,
        NOW(),
        NOW()
        ),
        (NULL,
        2,
        'YELLOW',
        'BAMBOO',
        'chenqw0621@edu.jlu.com',
        277,
        2,
        NOW(),
        NOW()
        );
```

Listing 162: 插入多个行的语句

19.3 插入检索出的数据

可以将另一个表中检索出来的数据插入进当前表:

```
INSERT INTO customer_copy1 (customer_id,
                            store_id,
                            first_name,
                            last_name,
                            email,
                            address_id,
                            active,
                            create_date,
                            last_update)
SELECT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       active,
       create_date,
       last_update
FROM customer;
```

Listing 163: 插入检索出数据的语句

Tips 19.3.1 INSERT SELECT 中的列名在插入中的作用

MySQL 不关心 SELECT 返回的列名。它使用的是列的位置，因此 SELECT 中的第一列（不管其列名）将用来填充表列中指定的第一个列，这对于从使用不同列名的表中导入数据是非常有用的。

20 更新和删除数据

20.1 更新数据

UPDATE 既可以更新表中特定行，也可以更新表中所有行，要注意 WHERE 子句。
例如，现在我要把 first_name 为 BLUE 的客户的邮箱更新了，可以参考以下语句：

```
UPDATE customer
SET email = 'chen1172305218@gmail.com'
WHERE customer_id = 603;
```

Listing 164: 更新记录的语句

下面演示更新多个列：

```
UPDATE customer
SET email = 'cqw1172305218@163.com',
    first_name = 'GREEN'
WHERE customer_id = 604;
```

Listing 165: 更新多个记录的语句

Tips 20.1.1 UPDATE 的注意事项

1. UPDATE 语句中可以使用子查询，使得能用 SELECT 语句检索出的数据更新列数据。
2. 如果用 UPDATE 语句更新多行，并且在更新这些行中的一行或多行时出一个现错误，则整个 UPDATE 操作被取消（错误发生前更新的所有行被恢复到它们原来的值）。为即使是发生错误，也继续进行更新，可使用 IGNORE 关键字。（UPDATE IGNORE customers）

删除某个列的值可以把他设为 NULL:

```
UPDATE customer
SET email = 'NULL'
WHERE customer_id = 604;
```

Listing 166: UPDATE 删除值的语句

20.2 删除数据

DELETE 不仅可以从表中删除特定的行，也可以从表中删除所有行，例如：

```
DELETE FROM customer
WHERE customer_id = 604;
```

Listing 167: 删除一行的语句

Tips 20.2.1 DELETE 的注意事项

1. DELETE 语句从表中删除行，但是 DELETE 不删除表本身。
2. 如果想从表中删除所有行，不要使用 DELETE，可使用 TRUNCATE TABLE 语句。
3. MySQL 没有撤销（undo）按钮。应该非常小心地使用 UPDATE 和 DELETE。