MySQL基础笔记

书栈(BookStack.CN)

目　录

致谢
介紹
1. 关于MySQL基础知识
2. MySQL用户有关命令
3. 检索数据
4. 高级数据过滤
5. 用通配符进行过滤
6. 创建计算字段
7. 使用数据处理函数
8. 汇总数据
9. 分组数据
10. 使用子查询
11. 联结表
12. 创建高级链接
13. 组合查询
14. 插入数据库
15. 更新和删除数据
16. 创建和操作表
17. 使用视图
18. 管理事务处理
19. 备份脚本

致谢

当前文档 《MySQL基础笔记》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-10-22。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址: http://www.bookstack.cn/books/mysql-basic-note

书栈官网: http://www.bookstack.cn

书栈开源: https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

介紹

MySQL基础笔记

记录学习mysql的笔记，涉及安装，配置，基础SQL语句，与Python链接开发等等。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/index.html*

1. 关于MySQL基础知识

关于MySQL基础知识

- 系统环境MAC OS 10.10
- MySQL版本，5.7.9
  Mac 安装mysql很简单，官网下载安装包，双击安装就可以了，有几个地方需要注意

在Mac下用DMG包新安装mysql，在安装完毕最后一步会随机分配一个root密码，记住密码，安装完毕用root登录，但密码是过期状态，mysql默认情况下密码有效期是360天，需要重新改下，具体请看Password Expiration Policy.

基础的SQL语句主要来源于SQL in 10 Minutes, Sams Teach Yourself 这本书，这本书提供练习的表格，表格内容下载

基础的概念

- 数据库(database)：保存有组织的数据库。
- 表(table) ：特定类型的数据结果化清单。

存储在表中的数据是一种类型的数据或者一个清单的数据

- 模式(schema) 关于数据库和表的布局及特性
- 列(colomn) 表中的一个字段，所以表的不由一个或者多个列组成。

理解列最后的办法，是想象成一个网络，网络中每一列存储着一个特定信息。

- 数据类型(datatype) 所容许的数据类型，每个类别都有相应的数据类型，用来限制该列中允许的类型
- 行(row)：表中的数据是按行存储的，把表想象成网络，垂直的列，水平为行。
- 主键(primary key)：表中每一行都应该有可以唯一标识自己的列，一列中唯一能够识别表中的每一行的值。在设计表是应该总是设计带有主键，设置为主键瞒着的条件为：
  - 任意两行都带有相同的值
  - 每个行都必须带有主键值
  - 主键列中的值不允许修改更新
  - 主键值不能重用。

样例表

创建一个练习用的样例表格，一共5张表，表的内容用途

- 管理供应商
- 管理产品目录
- 管理客户列表
- 录入客户订单

新建一个数据库

```
mysql> CREATE DATABASE SQL_Learning;
Query OK, 1 row affected (0.01 sec)
```

新建 `Vendors` 表存储卖产品的供应商。

先切换到该数据库

```
mysql> USE SQL_Learning;
Database changed

mysql> CREATE TABLE Vendors
    -> (
    ->   vend_id      char(10) NOT NULL ,
    ->   vend_name    char(50) NOT NULL ,
    ->   vend_address char(50) NULL ,
    ->   vend_city    char(50) NULL ,
    ->   vend_state   char(5)  NULL ,
    ->   vend_zip     char(10) NULL ,
    ->   vend_country char(50) NULL
    -> );
Query OK, 0 rows affected (0.08 sec)
```

- NOT NULL表示值不能为空
- char(10) 保存固定长度的字符串，这里指定10个字符串
- vend_id 主键

Products 表包含产品的目录

```
mysql> CREATE TABLE Products
    -> (
    ->   prod_id    char(10)      NOT NULL ,
    ->   vend_id    char(10)      NOT NULL ,
    ->   prod_name  char(255)     NOT NULL ,
```

```
    ->   prod_price decimal(8,2)  NOT NULL ,
    ->   prod_desc  text          NULL
    -> );
Query OK, 0 rows affected (0.03 sec)
```

- prod_id 主键
- vend_id 外键，与供应商ID关联
- decimal(8,2) 作为字符串存储的 DOUBLE 类型，允许固定的小数点。
  创建Customers 表

```
mysql> CREATE TABLE Customers
    -> (
    ->   cust_id      char(10)  NOT NULL ,
    ->   cust_name    char(50)  NOT NULL ,
    ->   cust_address char(50)  NULL ,
    ->   cust_city    char(50)  NULL ,
    ->   cust_state   char(5)   NULL ,
    ->   cust_zip     char(10)  NULL ,
    ->   cust_country char(50)  NULL ,
    ->   cust_contact char(50)  NULL ,
    ->   cust_email   char(255) NULL
    -> );
Query OK, 0 rows affected (0.05 sec)
```

- cust_id 主键
  创建Orders表，一行一个产品

```
mysql> CREATE TABLE Orders
    -> (
    ->   order_num  int      NOT NULL ,
    ->   order_date datetime NOT NULL ,
    ->   cust_id    char(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)
```

- order_num 主键
- cust_id 外键，关联Customers ID
  OrderItems表 存储每个订单的实际物品。

```
mysql> CREATE TABLE OrderItems
    -> (
```

```
3.      ->   order_num  int          NOT NULL ,
4.      ->   order_item int          NOT NULL ,
5.      ->   prod_id    char(10)     NOT NULL ,
6.      ->   quantity   int          NOT NULL ,
7.      ->   item_price decimal(8,2) NOT NULL
8.      -> );
9.  Query OK, 0 rows affected (0.02 sec)
```

- order_num 订单号，关联到Orders 表的order_num
- order_item 订单物品号，
- prod_id 产品ID
- quantity 产品数量
- item_price 产品的价格
  创建`primary key` 主键

```
1.  ALTER TABLE Customers ADD PRIMARY KEY (cust_id);
2.  ALTER TABLE OrderItems ADD PRIMARY KEY (order_num, order_item);
3.  ALTER TABLE Orders ADD PRIMARY KEY (order_num);
4.  ALTER TABLE Products ADD PRIMARY KEY (prod_id);
5.  ALTER TABLE Vendors ADD PRIMARY KEY (vend_id);
```

创建 `foreign key` 外键，指向另一个表 `PRIMARY KEY` 主键

```
1.  ALTER TABLE OrderItems ADD CONSTRAINT FK_OrderItems_Orders FOREIGN KEY
    (order_num) REFERENCES Orders (order_num);
```

- 把OrderItems 表中的order_num关联到Orders表中的order_num

  ```
  1.  ALTER TABLE OrderItems ADD CONSTRAINT FK_OrderItems_Products FOREIGN KEY
      (prod_id) REFERENCES Products (prod_id);
  ```

- 把OrderItems 表中的prod_id关联到Products 表中的prod_id

  ```
  1.  ALTER TABLE Orders ADD CONSTRAINT FK_Orders_Customers FOREIGN KEY
      (cust_id) REFERENCES Customers (cust_id);
  ```

- 把Orders 表中的cust_id 关联到Customers 表中的cust_id

  ```
  1.  ALTER TABLE Products ADD CONSTRAINT FK_Products_Vendors FOREIGN KEY
      (vend_id) REFERENCES Vendors (vend_id);
  ```

- 把Products 表中的vend_id 关联到Vendors中的vend_id```
  表插入表数据直接下载表格的数据，一个表一个表批量插入就行。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter1.html*

2. MySQL用户有关命令

MySQL用户有关命令

进入

```
mysql -u usename -p password -P 默认3306
mysql -S /tmp/mysql.sock -uroot -h192.168.56.1 -P3306 -p1234567 指定sock登录
\h 获取帮助
\q 退出 or quit
```

修改msyql密码

```
$ mysql -u root
mysql> USE mysql;
mysql> UPDATE user SET authentication_string=PASSWORD("NEWPASSWORD") WHERE
User='root';
```

创建用户

```
mysql> CREATE USER 'test1'@'localhost' identified by '1234567';
Query OK, 0 rows affected (0.00 sec)
```

查询用户

```
mysql> SELECT USER FROM mysql.user; 查询所有用户
mysql> SHOW GRANTS For root@'localhost'; 查询具体某个用户
```

删除用户

```
mysql> DROP USER 'test1'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

GRANT语句授权用户登录

```
mysql> GRANT ALL ON *.* TO 'test1'@'192.168.56.1' IDENTIFIED BY '1234567';
```

```
2.  Query OK, 0 rows affected, 1 warning (0.01 sec)
```

- 限定IP地址 192.168.56.1登录操作
- *. 第一个表示所有数据库，第二个表示所有表
- 最后是远程密码
  具体指定用户可用的语句，限制test1只能用SELECT语句。

```
1.  mysql> GRANT SELECT ON *.* TO 'test1'@'localhost' identified BY '1234567';
2.  Query OK, 0 rows affected, 1 warning (0.00 sec)
3.
4.  mysql> flush privileges; 刷新权限
5.  Query OK, 0 rows affected (0.00 sec)
```

查询是否授权成功

```
1.  mysql> SELECT * FROM USER WHERE HOST='192.168.56.1'\G;
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter2.html*

3. 检索数据

基础SQL语句-检索数据

SELECT 语句

是最常用的SQL语句了，用来索引一个或者多个表信息。

- 关键字(keyword)
  作为SQL组成部分的字段，关键字不能作为表或者列的名字。

使用SELECT索引数据，必须至少给出两条信息， 想要什么？ 从什么地方获取？

检查单个列

```
1.  SELECT prod_name FROM Products;
```

解释：使用SELECT 语句从 Products 表中检索一个名为prod_name 的列，FROM 关键字从指定的标名索引。

输出结果

```
1.  mysql> SELECT prod_name FROM Products;
2.  +----------------------+
3.  | prod_name            |
4.  +----------------------+
5.  | Fish bean bag toy    |
6.  | Bird bean bag toy    |
7.  | Rabbit bean bag toy  |
8.  | 8 inch teddy bear    |
9.  | 12 inch teddy bear   |
10. | 18 inch teddy bear   |
11. | Raggedy Ann          |
12. | King doll            |
13. | Queen doll           |
14. +----------------------+
15. 9 rows in set (0.00 sec)
```

- SQL语句分成多好容易阅读与调试，如果语句较长
- SQL语句必须以(;)结束。

- SQL语句不区分大小写，除了表名，跟值以外，SQL关键字使用大写，便于阅读

索引多个列

与索引单列对比，唯一的不同是必须，在SELECT 关键字后给出多个列名，列名直接用 `,` 隔开，最后一列不需要。

```
1.  SELECT prod_id, prod_name, prod_price FROM Products;
```

解释：使用SELECT 从表Products 中选择数据，指定3个列名，prod_id, prod_name, prod_price

输出：

```
1.  mysql> SELECT prod_id, prod_name, prod_price FROM Products;
2.  +---------+---------------------+------------+
3.  | prod_id | prod_name           | prod_price |
4.  +---------+---------------------+------------+
5.  | BNBG01  | Fish bean bag toy   |       3.49 |
6.  | BNBG02  | Bird bean bag toy   |       3.49 |
7.  | BNBG03  | Rabbit bean bag toy |       3.49 |
8.  | BR01    | 8 inch teddy bear   |       5.99 |
9.  | BR02    | 12 inch teddy bear  |       8.99 |
10. | BR03    | 18 inch teddy bear  |      11.99 |
11. | RGAN01  | Raggedy Ann         |       4.99 |
12. | RYL01   | King doll           |       9.49 |
13. | RYL02   | Queen doll          |       9.49 |
14. +---------+---------------------+------------+
15. 9 rows in set (0.01 sec)
```

检索所有列

```
1.  SELECT * FROM Products;
```

使用通配符 * 表示返回表中所有的列

```
1.  mysql> SELECT * FROM Products;
2.  +---------+---------+---------------------+------------+----------------------
    ----------------------------------------------+
3.  | prod_id | vend_id | prod_name           | prod_price | prod_desc
    |
```

```
+---------+---------+--------------------+------------+-----------------------------------------------+
| BNBG01  | DLL01   | Fish bean bag toy  |       3.49 | Fish bean bag toy, complete with bean bag worms with which to feed it |
| BNBG02  | DLL01   | Bird bean bag toy  |       3.49 | Bird bean bag toy, eggs are not included      |
| BNBG03  | DLL01   | Rabbit bean bag toy |       3.49 | Rabbit bean bag toy, comes with bean bag carrots |
| BR01    | BRS01   | 8 inch teddy bear  |       5.99 | 8 inch teddy bear, comes with cap and jacket  |
| BR02    | BRS01   | 12 inch teddy bear |       8.99 | 12 inch teddy bear, comes with cap and jacket |
| BR03    | BRS01   | 18 inch teddy bear |      11.99 | 18 inch teddy bear, comes with cap and jacket |
| RGAN01  | DLL01   | Raggedy Ann        |       4.99 | 18 inch Raggedy Ann doll                      |
| RYL01   | FNG01   | King doll          |       9.49 | 12 inch king doll with royal garments and crown |
| RYL02   | FNG01   | Queen doll         |       9.49 | 12 inch queen doll with royal garments and crown |
+---------+---------+--------------------+------------+-----------------------------------------------+
9 rows in set (0.00 sec)
```

- 除非需要表中每一列，或者不明确指定列，否则不要使用* 通配符。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter3.html*

4. 高级数据过滤

高级数据过滤

操作符(operator)

用来改变WHERE子句中的子句关键字，也成逻辑操作符。

- AND操作符
  通过使用AND来给WHERE子句附加条件。

索引出供应商'DLL01'制造且价格小于等于4美金的所有产品名称和价格。

```
1.
2.  mysql> SELECT prod_id, prod_price, prod_name
3.      -> FROM Products
4.      -> WHERE vend_id = 'DLL01' AND prod_price <= 4;
5.  +---------+------------+--------------------+
6.  | prod_id | prod_price | prod_name          |
7.  +---------+------------+--------------------+
8.  | BNBG01  |       3.49 | Fish bean bag toy  |
9.  | BNBG02  |       3.49 | Bird bean bag toy  |
10. | BNBG03  |       3.49 | Rabbit bean bag toy |
11. +---------+------------+--------------------+
12. 3 rows in set (0.02 sec)
```

解释: SLELECT 语句中的子句WHERE包含两个条件，供应商指定DLL01,价格高于4美金，不显示，如果价格小于 4美金，都不术语DELL01的，也不显示。

OR操作符

检索匹配任意条件。

```
1.  mysql> SELECT prod_name, prod_price
2.      -> FROM Products
3.      -> WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';
4.  +--------------------+------------+
5.  | prod_name          | prod_price |
6.  +--------------------+------------+
7.  | Fish bean bag toy  |       3.49 |
8.  | Bird bean bag toy  |       3.49 |
```

```
| Rabbit bean bag toy |       3.49 |
| 8 inch teddy bear   |       5.99 |
| 12 inch teddy bear  |       8.99 |
| 18 inch teddy bear  |      11.99 |
| Raggedy Ann         |       4.99 |
+---------------------+------------+
7 rows in set (0.01 sec)
```

解释：索引供应商所有产品的产品名和价格，并匹配任意条件 DLL01或者BRS01.

计算次序

WHERE 运行AND 与 RO 结合，进行复杂操作，和高级过滤。

检索10美金以上，并且由DLL10或者BRSO1制造。

```
mysql> SELECT prod_name, prod_price FROM Products WHERE vend_id = 'DLL01' OR
vend_id = 'BRS01' AND prod_price >= 10;
+---------------------+------------+
| prod_name           | prod_price |
+---------------------+------------+
| Fish bean bag toy   |       3.49 |
| Bird bean bag toy   |       3.49 |
| Rabbit bean bag toy |       3.49 |
| 18 inch teddy bear  |      11.99 |
| Raggedy Ann         |       4.99 |
+---------------------+------------+
5 rows in set (0.00 sec)
```

返回的价格带有10美金一下的， 原因是AND有优先级 ，SQL在处理 OR前，先处理了AND,直接检索BRS01,或者DLL01，而忽略了价格。

解决的方法是用 园括号进行分组操作。

```
mysql> SELECT prod_name, prod_price
    -> FROM Products
    -> WHERE (vend_id = 'DLL01' OR vend_id = 'BRS01')
    -> AND prod_price >= 10;
+--------------------+------------+
| prod_name          | prod_price |
+--------------------+------------+
| 18 inch teddy bear |      11.99 |
+--------------------+------------+
```

```
10. 1 row in set (0.01 sec)
```

()圆括号具有比AND，RO更高的操作计算顺序。

注意： 使用AND 和OR操作WHERE句子，都应该用圆括号明确分组操作。

IN 操作符

IN操作符用来指定范围，范围中的每一条，都进行匹配。IN取值规律，由逗号分割，全部放置括号中。

```
1.  mysql> SELECT prod_name, prod_price
2.      -> FROM Products
3.      -> WHERE vend_id IN ('DLL01', 'BRS01')
4.      -> ORDER BY prod_name;
5.  +--------------------+------------+
6.  | prod_name          | prod_price |
7.  +--------------------+------------+
8.  | 12 inch teddy bear |       8.99 |
9.  | 18 inch teddy bear |      11.99 |
10. | 8 inch teddy bear  |       5.99 |
11. | Bird bean bag toy  |       3.49 |
12. | Fish bean bag toy  |       3.49 |
13. | Rabbit bean bag toy |       3.49 |
14. | Raggedy Ann        |       4.99 |
15. +--------------------+------------+
16. 7 rows in set (0.01 sec)
```

解释： 用SELECT检索，DLL01和BRS01制造的所有产品，IN操作符后跟由逗号分割的合法值清单。

IN 相当与完成了OR相同的功能，下面的结果与上面输出结果一样

```
1.  mysql> SELECT prod_name, prod_price
2.      -> FROM Products
3.      -> WHERE vend_id = 'DLL01' OR vend_id = 'BRS01'
4.      -> ORDER BY prod_name;
5.  +--------------------+------------+
6.  | prod_name          | prod_price |
7.  +--------------------+------------+
8.  | 12 inch teddy bear |       8.99 |
9.  | 18 inch teddy bear |      11.99 |
10. | 8 inch teddy bear  |       5.99 |
11. | Bird bean bag toy  |       3.49 |
12. | Fish bean bag toy  |       3.49 |
```

```
| Rabbit bean bag toy |       3.49 |
| Raggedy Ann         |       4.99 |
+---------------------+------------+
7 rows in set (0.00 sec)
```

- 使用IN的优点:
  - 语法清晰，特别是语法较长时
  - 操作符少，计算次序容易管理
  - IN比OR执行速度快
  - 最大的优点，可以包含其他SELECT语句，能够更加动态的建立WHERE子句。

NOT操作符

NOT操作符总是与其他操作符一起使用，用在要过滤的前面。

```
mysql> SELECT vend_id, prod_name FROM Products WHERE NOT vend_id = 'DLL01'
ORDER BY prod_name;
+---------+--------------------+
| vend_id | prod_name          |
+---------+--------------------+
| BRS01   | 12 inch teddy bear |
| BRS01   | 18 inch teddy bear |
| BRS01   | 8 inch teddy bear  |
| FNG01   | King doll          |
| FNG01   | Queen doll         |
+---------+--------------------+
5 rows in set (0.00 sec)
```

列出不带有DLL01之外的所有产品。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter4.html*

5. 用通配符进行过滤

用通配符进行过滤

利用通配符进行复杂的数据操作。

LIKE 与 REGEXP 操作符

当需要搜索产品文本中包含某个特定关键字的所有产品，使用通配符来创建比较特定的数据搜索模式。

- 通配符(wildcard) 用来匹配值的一部分特殊字符。
- 搜索模式(search pattern) 由字母值，通配符两租组合构成的搜索条件。

通配符是SQL的WHERE子句中的特殊含义字符，子句中使用通配符必须使用LIKE操作符。

百分号%通配符

表示任何符合出现任意次数。

```
mysql> SELECT prod_id, prod_name
    -> FROM Products
    -> WHERE prod_name LIKE 'Fish%';
+---------+--------------------+
| prod_id | prod_name          |
+---------+--------------------+
| BNBG01  | Fish bean bag toy |
+---------+--------------------+
1 row in set (0.01 sec)
```

检索以Fish 开头的词汇，Fish之后任意词汇，区分大小写。

```
mysql> SELECT prod_id, prod_name
    -> FROM Products
    -> WHERE prod_name LIKE '%bean bag%';
+---------+---------------------+
| prod_id | prod_name           |
+---------+---------------------+
| BNBG01  | Fish bean bag toy   |
| BNBG02  | Bird bean bag toy   |
| BNBG03  | Rabbit bean bag toy |
```

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

匹配任意位置包含 bean bag的值

```
mysql> SELECT prod_name
    -> FROM Products
    -> WHERE prod_name LIKE 'F%y';
+--------------------+
| prod_name          |
+--------------------+
| Fish bean bag toy |
+--------------------+
1 row in set (0.00 sec)
```

匹配F开头，y结尾的所以产品

```
mysql> SELECT prod_name
    -> FROM Products
    -> WHERE prod_name LIKE 'F%y';
+--------------------+
| prod_name          |
+--------------------+
| Fish bean bag toy |
+--------------------+
1 row in set (0.00 sec)
```

下划线 _ 通配符

下划线与%不同的是匹配单个字符，而不是多个字符。

```
mysql> SELECT prod_id, prod_name
    -> FROM Products
    -> WHERE prod_name LIKE '_ inch teddy bear';
+---------+--------------------+
| prod_id | prod_name          |
+---------+--------------------+
| BR01    | 8 inch teddy bear |
+---------+--------------------+
1 row in set (0.00 sec)
```

一个_匹配一个字符串。

方括号[]通配符

匹配任意带有JM的字符串的列。

```
mysql> SELECT cust_contact FROM Customers WHERE cust_contact REGEXP '[JM]'
ORDER BY cust_contact;
+----------------+
| cust_contact   |
+----------------+
| Jim Jones      |
| John Smith     |
| Kim Howard     |
| Michelle Green |
+----------------+
4 rows in set (0.00 sec)
```

匹配开头为J的列

```
mysql> SELECT cust_contact  FROM Customers WHERE cust_contact REGEXP '[J%]'
ORDER BY cust_contact;
+--------------+
| cust_contact |
+--------------+
| Jim Jones    |
| John Smith   |
+--------------+
2 rows in set (0.01 sec)
```

通配符技巧

- 其他操作如果能达到相同的效果，就不要用通配符。
- 使用通配符尽量，缩小检索范围。
- 主要通配符的位置

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter5.html*

6. 创建计算字段

第七章 创建计算字段

如何创建计算字段以及怎么样从应用程序中使用别名。

计算字段

存储在数据库表中的数据一般不是应用程序所需要的格式，例如：

- 显示两个信息，但不是在用一个表
- 不同列中，但程序需要把他们作为一个格式的字段检索出来
- 列数据是大小混合，但程序需要把所以数据按大写表示。
- 物品订单表存储的物品的价格和数量，但没有存储物品的总价，打印时，需要物品的总价格。
- 根据需要表的数据进行总数，平均数等计算。
  上面的情况都我们需要从数据库中转换，计算格式化，而不是索引出来再进行计算。

计算字段是在运行时SELECT语句内创建的。

字段(field)

与列(column)意思类似，经常相互转换使用，字段通常用在计算字段的链接上。

在SQL内完成转换和格式化，比在客户机应用程序内完成，处理数度更快。

拼接字段

例子：

vendors 表包含供应商id 跟 位置信息，现在需要生成一个供应商表，需要格式化名称，列出供应商位置。此报表需要单个值，而表中的数据存储在， `vend_name` 和 `vend_country` 中，还需要建 `vend_country` 括起来。

拼接(concatenate)

将值联结到一起创建单个值。

在SQL中使用一个特使操作符来拼接两个列，+操作符用加号(+),两个竖杆(||)表示。

在mysql使用CONCAT()函数把项表链接起来，而|| 通等于操作符OR 而&&通等于AND操作符。

下面是在mysql中执行的结果：

```
mysql> SELECT CONCAT(vend_name, ' (', vend_country, ')')
    -> FROM Vendors
    -> ORDER BY vend_name;
+-------------------------------------------+
| CONCAT(vend_name, ' (', vend_country, ')') |
+-------------------------------------------+
| Bear Emporium (USA)                       |
| Bears R Us (USA)                          |
| Doll House Inc. (USA)                     |
| Fun and Games (England)                   |
| Furball Inc. (USA)                        |
| Jouets et ours (France)                   |
+-------------------------------------------+
6 rows in set (0.01 sec)
```

解释：

- 存储vend_name列中的名字
- vend_name 隔一个空格加一个(圆括号
- 存储在vend_country列中的国家
- 包含一个闭圆括号串。

别名

拼接的地址字段，没有一个名字，无法给客户机应用，所以需要字段 `别名(alias)` ,另一种叫法导出列 `(derived column)`

别名可以用 `AS关键字赋予` 。

```
mysql> SELECT vend_name, CONCAT(vend_name, '(', vend_country, ')') AS
vend_titel
    -> FROM Vendors
    -> ORDER BY vend_name;
+----------------+------------------------+
| vend_name      | vend_titel             |
+----------------+------------------------+
| Bear Emporium  | Bear Emporium(USA)     |
| Bears R Us     | Bears R Us(USA)        |
| Doll House Inc.| Doll House Inc.(USA)   |
| Fun and Games  | Fun and Games(England) |
| Furball Inc.   | Furball Inc.(USA)      |
| Jouets et ours | Jouets et ours(France) |
+----------------+------------------------+
```

```
14. 6 rows in set (0.00 sec)
```

解释：这样用AS `vend_title` 指定拼接输出结果的列名，

别名的另一个用法，列重命名

```
1.  mysql> SELECT vend_name  AS VEND_TEST FROM Vendors ;
2.  +-----------------+
3.  | VEND_TEST       |
4.  +-----------------+
5.  | Bear Emporium   |
6.  | Bears R Us      |
7.  | Doll House Inc. |
8.  | Fun and Games   |
9.  | Furball Inc.    |
10. | Jouets et ours  |
11. +-----------------+
12. 6 rows in set (0.00 sec)
```

把 `vend_name` 输出重命名为 `VEND_TEST` .

执行算术计算

用于检索出数据进行计算。

检索出Orders 表中包含收到的所有订单，OrderItems表包含每个订单中的各项物品，

```
1.  mysql> SELECT prod_id, quantity, item_price FROM OrderItems WHERE order_num =
    20008;
2.  +---------+----------+------------+
3.  | prod_id | quantity | item_price |
4.  +---------+----------+------------+
5.  | RGAN01  |        5 |       4.99 |
6.  | BR03    |        5 |      11.99 |
7.  | BNBG01  |       10 |       3.49 |
8.  | BNBG02  |       10 |       3.49 |
9.  | BNBG03  |       10 |       3.49 |
10. +---------+----------+------------+
11. 5 rows in set (0.00 sec)BG03   |       3.49 |
12. +----------+------------+
13. 5 rows in set (0.03 sec)
```

解释：检索 `prod_id` ， `quantity` ， `item_price` 中 订单号码为20008的所有物品。

总汇出物品的价格，item_price 包含每项物品的单价，quantity包含数量。

```
mysql> SELECT prod_id, quantity, item_price,
    -> quantity*item_price AS expanded_price
    -> FROM OrderItems
    -> WHERE order_num = 20008;
+---------+----------+------------+----------------+
| prod_id | quantity | item_price | expanded_price |
+---------+----------+------------+----------------+
| RGAN01  |        5 |       4.99 |          24.95 |
| BR03    |        5 |      11.99 |          59.95 |
| BNBG01  |       10 |       3.49 |          34.90 |
| BNBG02  |       10 |       3.49 |          34.90 |
| BNBG03  |       10 |       3.49 |          34.90 |
+---------+----------+------------+----------------+
5 rows in set (0.01 sec)
```

解释： `quantity*item_price` ，单价乘以数量，输出 `expanded_price` 列为计算字段，客户端使用这个列，就跟使用其他列一样。

支持的运算字符 `+ 加, - 减, * 乘, 除 /`

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter6.html*

7. 使用数据处理函数

使用数据处理函数

关于函数使用，与带来的问题。

函数

函数主要给数据提供处理与转换方便。

大多数SQL实现的函数

- 用于处理文本串(删除，充值，大小写转换)
- 用于在数值的数据上进行算术(返回绝对值，代数运算)操作。
- 用于处理日期时间值并从这些值中提取特定成份。
- 返回DBMS正使用的特殊信息(用户登录信息)。

文本处理函数

使用UPPER()函数来转换大小写。

```
mysql> SELECT vend_name, UPPER(vend_name) AS
    -> vend_name_upcase
    -> FROM Vendors
    -> ORDER BY vend_name;
+----------------+------------------+
| vend_name      | vend_name_upcase |
+----------------+------------------+
| Bear Emporium  | BEAR EMPORIUM    |
| Bears R Us     | BEARS R US       |
| Doll House Inc. | DOLL HOUSE INC.  |
| Fun and Games  | FUN AND GAMES    |
| Furball Inc.   | FURBALL INC.     |
| Jouets et ours | JOUETS ET OURS   |
+----------------+------------------+
6 rows in set (0.01 sec)
```

解释：vend_name 列出两次，一次储存值，一次将文本转换成大写。

常用处理文本函数

- LENGTH() 返回串长度。

```
mysql> SELECT vend_name, LENGTH(vend_name) AS vend_name_length
  -> FROM Vendors
  -> ORDER BY vend_name;
+----------------+------------------+
| vend_name      | vend_name_length |
+----------------+------------------+
| Bear Emporium  |               13 |
| Bears R Us     |               10 |
| Doll House Inc. |               15 |
| Fun and Games  |               13 |
| Furball Inc.   |               12 |
| Jouets et ours |               14 |
+----------------+------------------+
6 rows in set (0.01 sec)
```

- LOWER() 转换小写

```
mysql> SELECT vend_name, LOWER(vend_name)
  -> AS vend_name_lower
  -> FROM Vendors
  -> ORDER BY vend_name;
+----------------+----------------+
| vend_name      | vend_name_lower |
+----------------+----------------+
| Bear Emporium  | bear emporium  |
| Bears R Us     | bears r us     |
| Doll House Inc. | doll house inc. |
| Fun and Games  | fun and games  |
| Furball Inc.   | furball inc.   |
| Jouets et ours | jouets et ours |
+----------------+----------------+
6 rows in set (0.00 sec)
```

- 返回串中SOUNDEX值，意思是建任何文本串转为表述其语言表述的字母数字模式的算法。

```
mysql> SELECT vend_name, SOUNDEX(vend_name)
  -> AS vend_name_soundex
  -> FROM Vendors
  -> ORDER BY vend_name;
+----------------+------------------+
```

```
| vend_name       | vend_name_soundex |
+-----------------+-------------------+
| Bear Emporium   | B65165            |
| Bears R Us      | B6262             |
| Doll House Inc. | D4252             |
| Fun and Games   | F53252            |
| Furball Inc.    | F61452            |
| Jouets et ours  | J32362            |
+-----------------+-------------------+
```

例子:

Customers 表有一个名为Kids Place,顾客为Michael Green, 通过Michelle Green 类似发音来找到。

```
mysql> SELECT cust_name, cust_contact
    -> FROM Customers
    -> WHERE SOUNDEX(cust_contact) = SOUNDEX('Michael Green');
+------------+----------------+
| cust_name  | cust_contact   |
+------------+----------------+
| Kids Place | Michelle Green |
+------------+----------------+
1 row in set (0.03 sec)
```

SOUNDEX() 函数转换 Michael Green,ichelle Green值，两个发音类似，所以能检索出来。

日期和时间处理函数

存储为特殊的数据类型，主要用于排序过滤。

更多的时间函数Date and Time Functions

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter7.html*

8. 汇总数据

汇总数据

关于如何利用函数汇总表的数据。

聚集函数

- 确定表中的行数
- 获得表中行组的和
- 找出表列(所有行，特定行)的最大，最小，平均值。
  上面的例子需要对表中的数据汇总，而不是实际数据本身，所以可以不需要返回时间数据，浪费资源

聚集函数(aggregate function)

运行在行组上，计算和返回单个值的函数。

- AVG()：返回所有列或者某个列平均值。
  计算表中的行数并计算特定列值之和，求得改列的平均值。

```
mysql> SELECT AVG(prod_price) AS avg_price
    -> FROM Products;
+-----------+
| avg_price |
+-----------+
|  6.823333 |
+-----------+
1 row in set (0.01 sec)
```

解释：计算Products表中所以产品的平均价格。

计算特定行的平均值，

```
mysql> SELECT AVG(prod_price) AS avg_price
    -> FROM Products
    -> WHERE vend_id = 'DLL01';
+-----------+
| avg_price |
+-----------+
```

```
| 3.865000 |
+----------+
1 row in set (0.01 sec)
```

解释：WHERE 子句过滤出DELL01平均值，并返回该供应商产品的平均值。

- COUNT()函数计算表中行的数目或符合特定条件的涵数目。
  - 忽略表列中包含的空值(NULL)与非空值，对表中数目进行计算。
  - 使用COUNT(column) 对特定列中具有值的行进行计算，忽略NULL值。

```
mysql> SELECT COUNT(*) AS num_cust
-> FROM Customers;
+----------+
| num_cust |
+----------+
|        5 |
+----------+
1 row in set (0.01 sec)
```

返回custoemrs 中客户的总数，不管行中各列的数值。

```
mysql> SELECT COUNT(cust_email) AS num_cust
    -> FROM Customers;
+----------+
| num_cust |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

值返回有email地址的客户计数，结果为3，表述只有3个客户有电子邮件地址。

MAX()返回指定的列中最大的值

```
mysql> SELECT MAX(prod_price) AS max_price
    -> FROM Products;
+-----------+
| max_price |
+-----------+
|     11.99 |
+-----------+
1 row in set (0.01 sec)
```

解释：返回Products表中最贵的物品价格。

- MIN() 返回最小值

```
mysql> SELECT MIN(prod_price) AS min_price
  -> FROM Products;
+-----------+
| min_price |
+-----------+
|      3.49 |
+-----------+
1 row in set (0.00 sec)
```

- SUM()返回指定的列值的总和

```

mysql> SELECT SUM(quantity) AS items_ordered
  -> FROM OrderItems
  -> WHERE order_num = 20005;
+---------------+
| items_ordered |
+---------------+
|           200 |
+---------------+
1 row in set (0.01 sec)
```

解释：返回计算quantity 值之和，WHERE子句限制值统计某个订单的值。

用SUM()组合计算值

```
mysql> SELECT SUM(item_price*quantity) AS total_price
    -> FROM OrderItems
    -> WHERE order_num = 20005;
+-------------+
| total_price |
+-------------+
|     1648.00 |
+-------------+
1 row in set (0.01 sec)
```

解释：合计所以订单 item_price价格 乘以quantity数量之和的总数，WHERE子句某个订单物品。

聚集不同值

- 对所有的行执行计算，指定ALL参数或者不改参数(默认是ALL行为)
- 只包含不同的值，指定DISTINCT参数

```
mysql> SELECT AVG(DISTINCT prod_price) AS avg_price
  -> FROM Products
  -> WHERE vend_id = 'DLL01';
+-----------+
| avg_price |
+-----------+
|  4.240000 |
+-----------+
1 row in set (0.02 sec)
```

解释:与上一个例子不同的是，排除prod_price 中相同的值，只计算不同的值，数量少了所以平均值高了。

组合聚集函数

用SELECT 来组合聚集函数。

```
mysql> SELECT COUNT(*) AS num_items,
    -> MIN(prod_price) AS price_min,
    -> MAX(prod_price) AS price_max,
    -> AVG(prod_price) AS price_avg
    -> FROM Products;
+-----------+-----------+-----------+-----------+
| num_items | price_min | price_max | price_avg |
+-----------+-----------+-----------+-----------+
|         9 |      3.49 |     11.99 |  6.823333 |
+-----------+-----------+-----------+-----------+
1 row in set (0.00 sec)
```

- 聚集函数用来总汇数据， SQL支持5个聚集函数，计算速度比在客户端快多。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter8.html*

9. 分组数据

分组数据

用GROUP BY 跟 HAVING子句，分组数据来汇总表内容子集。

创建分组

分组在SELECT语句的GROUP BY子句中建立。

```
mysql> SELECT vend_id, COUNT(*) AS num_prods
    -> FROM Products
    -> GROUP BY vend_id;
+---------+-----------+
| vend_id | num_prods |
+---------+-----------+
| BRS01   |         3 |
| DLL01   |         4 |
| FNG01   |         2 |
+---------+-----------+
3 rows in set (0.01 sec)
```

解释： `SELECT` 语句指定两个列， `vend_id` 包含供应商ID，为 `num_prods` 计算字段结果， `GROUP BY` 子句指示 vend_id 排序并分组数据，

GROUP BY子句重要规则：

- 包含任意数目的列，
- 如果在GROUP BY 子句中套入分组，数据将会最后规定的分组上进行总汇。
- GROUP BY 子句中列出的没列都必须是检索的列，有效的表达式，不能聚集函数。
- 大多数SQL不允许GROUP BY 带有长度可变的数据类型(文本，备注型字段)
- 除聚集计算语句外，SELECT 语句中，每个列都必须在GROUP BY子句中给出。
- 如果分组带有NULL值，将作为一个分组返回，如果多个将成一组。
- GROUP BY 子句必须出现在WHERE子句之后，

过滤分组

过滤分组规定包含哪些分组，排除哪些分组，用 `HAVING` 子句，与WHERE子句类似，唯一差别的是WHERE用来过滤行，HAVING过滤分组。也可以说HAVING在数据分组后过滤，WHERE在数据分组前进行过滤。

HAVING 支持所有WHERE的操作符。

```
mysql> SELECT cust_id, COUNT(*) AS orders
    -> FROM  Orders
    -> GROUP BY cust_id
    -> HAVING COUNT(*) >= 2;
+------------+--------+
| cust_id    | orders |
+------------+--------+
| 1000000001 |      2 |
+------------+--------+
1 row in set (0.00 sec)
```

解释：过滤出两个以上订单的分组

WHERE与HAVING子句结合使用

```
mysql> SELECT vend_id, COUNT(*) AS num_prods
    -> FROM Products
    -> WHERE prod_price >= 4
    -> GROUP BY vend_id
    -> HAVING COUNT(*) >= 2;
+---------+-----------+
| vend_id | num_prods |
+---------+-----------+
| BRS01   |         3 |
| FNG01   |         2 |
+---------+-----------+
2 rows in set (0.00 sec)
```

解释：第一行使用聚集函数，WHERE子句过滤除所有 `prod_price` 少于4的行，按vend_id分组，HAVING子句过滤计数2以上分组。

去掉WHERE 过滤

```
mysql> SELECT vend_id, COUNT(*) AS num_prods FROM Products  GROUP BY vend_id
HAVING COUNT(*) >= 2;
+---------+-----------+
| vend_id | num_prods |
+---------+-----------+
| BRS01   |         3 |
| DLL01   |         4 |
```

```
| FNG01    |         2 |
+---------+-----------+
3 rows in set (0.01 sec)
```

过滤出销售产品在4个，且价格是4一下的。

分组和排序

GROUP BY 与 ORDER BY区别

- GROUP BY
    - 排序产生的输出
    - 任意列都可以使用
    - 可以选择是否与聚集函数一起使用
- ORDER BY
    - 分组行，输出可能不是分组循序
    - 只可能使用选择列或表达式，且必须使用每个列表达式
    - 如果与聚集函数一起用，则必须使用
      注意：不用依赖于GROUP BY 排序，应该使用GROUP BY 时，也该处ORDER BY子句。

检索除3个或以上的物品订单号与订购物品数目：

```
mysql> SELECT order_num, COUNT(*) AS items
    -> FROM OrderItems
    -> GROUP BY order_num
    -> HAVING COUNT(*) >= 3;
+-----------+-------+
| order_num | items |
+-----------+-------+
|     20006 |     3 |
|     20007 |     5 |
|     20008 |     5 |
|     20009 |     3 |
+-----------+-------+
4 rows in set (0.00 sec)
```

按订购物品数目排序输出。

```
mysql> SELECT order_num, COUNT(*) AS items
    -> FROM OrderItems
```

```
    -> GROUP BY order_num
    -> HAVING COUNT(*) >=3
    -> ORDER BY items, order_num;
+-----------+-------+
| order_num | items |
+-----------+-------+
|     20006 |     3 |
|     20009 |     3 |
|     20007 |     5 |
|     20008 |     5 |
+-----------+-------+
4 rows in set (0.00 sec)
```

解释: GROUP BY 子句用来分组数据， COUNT(*)函数返回订单中物品数目，HAVING 子句过滤数据，返回3个或3个以上的物品订单，ORDER BY最后排序输出。

SELECT子句顺序

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	仅从 表中选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅按组计算聚集使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter9.html*

10. 使用子查询

使用子查询

关于子查询使用

利用子查询进行过滤

列出够物品RGAN01的所有客户。

- 检索包含物品RGAN01的所有订单编号。
- 检索具有前一步骤列出的订单编号所有客户ID。
- 检索前一步骤返回的所有客户ID的客户信息。

```
mysql> SELECT order_num
  -> FROM OrderItems
  -> WHERE prod_id = 'RGAN01';
+-----------+
| order_num |
+-----------+
|     20007 |
|     20008 |
+-----------+
2 rows in set (0.00 sec)
```

解释：列出所有RGAN01订单物品

```
mysql> SELECT cust_id
    -> FROM Orders
    -> WHERE order_num IN (20007, 20008);
+------------+
| cust_id    |
+------------+
| 1000000004 |
| 1000000005 |
+------------+
2 rows in set (0.00 sec)
```

把上面两个查询组合成子查询

```
mysql> SELECT cust_id
    -> FROM Orders
    -> WHERE order_num IN (SELECT order_num
    -> FROM OrderItems
    -> WHERE prod_id = 'RGAN01');
+------------+
| cust_id    |
+------------+
| 1000000004 |
| 1000000005 |
+------------+
2 rows in set (0.00 sec)
```

- 子查询是从内向外处理
- 先执行 SELECT order_num FROM OrderItems WHERE prod_id = 'RGAN01'
- 把返回的订单号，20007，20008两个值以IN操作符用逗号格式传递给外部查询，
- 再用SELECT cust_id FROM orders WHERE order_num IN (20007,20008) 检索客户的ID

```
mysql> SELECT cust_name, cust_contact
    -> FROM Customers
    -> WHERE cust_id IN (SELECT cust_id
    -> FROM Orders
    -> WHERE order_num IN (SELECT order_num
    -> FROM OrderItems
    -> WHERE prod_id = 'RGAN01'));
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
2 rows in set (0.01 sec)
```

- 解释：
  - SELECT order_num FROM OrderItems WHERE prod_id = 'RGAN01' 查询返回订单列表
  - 把上面的查询结果，给予 SELECT cust_id FROM Orders， 返回客户ID
  - 拿到返回的客户ID给予外层WHERE子句查询。
    注意： 子查询只能返回单个列。

作为计算字段使用子查询

例子：计算Customers中每个客户订单总数。

- 从Customers 表中检索客户列表
- 对检索出来的每个客户，统计其在Orders表中的订单数目。
  单个客户查询

```
mysql> SELECT COUNT(*) AS orders
    -> FROM Orders
    -> WHERE cust_id = '1000000001';
+--------+
| orders |
+--------+
|      2 |
+--------+
1 row in set (0.00 sec)
```

对每个客户执行COUNT(*)

```
mysql> SELECT cust_name,
    -> cust_state,
    -> (SELECT COUNT(*)
    -> FROM Orders
    -> WHERE Orders.cust_id = Customers.cust_id) AS
    -> orders
    -> FROM Customers
    -> ORDER BY cust_name;
+---------------+------------+--------+
| cust_name     | cust_state | orders |
+---------------+------------+--------+
| Fun4All       | IN         |      1 |
| Fun4All       | AZ         |      1 |
| Kids Place    | OH         |      0 |
| The Toy Store | IL         |      1 |
| Village Toys  | MI         |      2 |
+---------------+------------+--------+
5 rows in set (0.01 sec)
```

解释：对Customers表返回三列，`cust_name`，`cust_state`，`orders`.orders 是计算字段，由 `(SELECT COUNT(*) FROM Orders WHERE Orders.cust_id = Customers.cust_id)` 建立。每检索一个客户，执行一次计算，

`Orders.cust_id = Customers.cust_id` ，其中的句号，表示指定限定表名跟列，如果不具体指定表名，列名，将返回Orders 表中的订单总数。如下：

```
mysql> SELECT cust_name,
    -> cust_state,
    -> (SELECT COUNT(*)
    -> FROM Orders
    -> WHERE cust_id = cust_id) AS orders
    -> FROM Customers
    -> ORDER BY cust_name;
+---------------+------------+--------+
| cust_name     | cust_state | orders |
+---------------+------------+--------+
| Fun4All       | IN         |      5 |
| Fun4All       | AZ         |      5 |
| Kids Place    | OH         |      5 |
| The Toy Store | IL         |      5 |
| Village Toys  | MI         |      5 |
+---------------+------------+--------+
5 rows in set (0.00 sec)
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter10.html*

11. 联结表

联结表

什么是链接，为什么要使用，如何使用。

关系表

例子： 一个包含目录的数据库，其中每种类型物品占用一行，每种物品要存储的信息包括产品描述和价格，以及生产该产品的供应商信息。

有一个供应商生产多种物品，何处存储供应商的信息(地址，电话等)，如何分开存储。

- 同一个供应商存储的信息都是相同的，每种产品重复信息，浪费空间
- 如果供应商信息改变，执行改一次。
- 重复数据，难保证每次储存信息一致，不一致信息难管理，利用。
  关系数据库设计：
- 避免相同数据出现多次
- 信息被分解成一种数据，一个表
- 各表通过某些常用值相互关联
  上面的例子，设计两个表，一个存储供应商信息，一个存储产品信息。
- Vendors 表包含所有供应商信息，供应商的primary key 唯一的标识值vend_id。
- Products表只存储产品信息，与供应商的primary key vend_id 表关联，利用供应商的ID从Vendors表中找出相应的供应商详细信息。
  这样的设计刚好符合上面3点。

关系数据库优点 `可伸缩性` (scale),能够适应不断增加的工作量。

使用联结的好处

分解多个表方便存储，方便处理，可伸缩性强。

使用链接可以用一条SELECT中关联多个表返回一组输出。

注意:在设计关系数据库，避免在另一个关系表中插入非法的ID，可以设置关系表中值，只出现合法的值

创建联结

链接多个表

```
mysql> SELECT vend_name, prod_name, prod_price
    -> FROM Vendors, Products
    -> WHERE Vendors.vend_id = Products.vend_id;
+----------------+--------------------+------------+
| vend_name      | prod_name          | prod_price |
+----------------+--------------------+------------+
| Bears R Us     | 8 inch teddy bear  |       5.99 |
| Bears R Us     | 12 inch teddy bear |       8.99 |
| Bears R Us     | 18 inch teddy bear |      11.99 |
| Doll House Inc. | Fish bean bag toy  |       3.49 |
| Doll House Inc. | Bird bean bag toy  |       3.49 |
| Doll House Inc. | Rabbit bean bag toy |       3.49 |
| Doll House Inc. | Raggedy Ann        |       4.99 |
| Fun and Games  | King doll          |       9.49 |
| Fun and Games  | Queen doll         |       9.49 |
+----------------+--------------------+------------+
9 rows in set (0.00 sec)
```

解释：

- SELECT vend_name, prod_name, prod_price 指定检索的列，prod_name, prod_price 在同一个表。vend_name 在另外一个表
- From 指定联结两个表Vendors, Products
- WHERE子句限定 Vendors.vend_id = Products.vend_id 完全限定名。

WHERE子句的重要

- 笛卡儿积(cartesian product) 由没有联结的条件表关系返回的结果，保证所有联结都有WHERE子句，否则返回比的数据会比想要的数据多很多。

```

mysql> SELECT vend_name, prod_name, prod_price
    -> FROM Vendors, Products;
+----------------+--------------------+------------+
| vend_name      | prod_name          | prod_price |
+----------------+--------------------+------------+
| Bear Emporium  | Fish bean bag toy  |       3.49 |
```

```
 8. | Bears R Us      | Fish bean bag toy   |       3.49 |
 9. | Doll House Inc. | Fish bean bag toy   |       3.49 |
10. | Fun and Games   | Fish bean bag toy   |       3.49 |
11. | Furball Inc.    | Fish bean bag toy   |       3.49 |
12. | Jouets et ours  | Fish bean bag toy   |       3.49 |
13. | Bear Emporium   | Bird bean bag toy   |       3.49 |
14. | Bears R Us      | Bird bean bag toy   |       3.49 |
15. | Doll House Inc. | Bird bean bag toy   |       3.49 |
16. | Fun and Games   | Bird bean bag toy   |       3.49 |
17. | Furball Inc.    | Bird bean bag toy   |       3.49 |
18. | Jouets et ours  | Bird bean bag toy   |       3.49 |
19. | Bear Emporium   | Rabbit bean bag toy |       3.49 |
20. | Bears R Us      | Rabbit bean bag toy |       3.49 |
21. | Doll House Inc. | Rabbit bean bag toy |       3.49 |
22. | Fun and Games   | Rabbit bean bag toy |       3.49 |
23. | Furball Inc.    | Rabbit bean bag toy |       3.49 |
24. | Jouets et ours  | Rabbit bean bag toy |       3.49 |
25. | Bear Emporium   | 8 inch teddy bear   |       5.99 |
26. | Bears R Us      | 8 inch teddy bear   |       5.99 |
27. | Doll House Inc. | 8 inch teddy bear   |       5.99 |
28. | Fun and Games   | 8 inch teddy bear   |       5.99 |
29. | Furball Inc.    | 8 inch teddy bear   |       5.99 |
30. | Jouets et ours  | 8 inch teddy bear   |       5.99 |
31. | Bear Emporium   | 12 inch teddy bear  |       8.99 |
32. | Bears R Us      | 12 inch teddy bear  |       8.99 |
33. | Doll House Inc. | 12 inch teddy bear  |       8.99 |
34. | Fun and Games   | 12 inch teddy bear  |       8.99 |
35. | Furball Inc.    | 12 inch teddy bear  |       8.99 |
36. | Jouets et ours  | 12 inch teddy bear  |       8.99 |
37. | Bear Emporium   | 18 inch teddy bear  |      11.99 |
38. | Bears R Us      | 18 inch teddy bear  |      11.99 |
39. | Doll House Inc. | 18 inch teddy bear  |      11.99 |
40. | Fun and Games   | 18 inch teddy bear  |      11.99 |
41. | Furball Inc.    | 18 inch teddy bear  |      11.99 |
42. | Jouets et ours  | 18 inch teddy bear  |      11.99 |
43. | Bear Emporium   | Raggedy Ann         |       4.99 |
44. | Bears R Us      | Raggedy Ann         |       4.99 |
45. | Doll House Inc. | Raggedy Ann         |       4.99 |
46. | Fun and Games   | Raggedy Ann         |       4.99 |
47. | Furball Inc.    | Raggedy Ann         |       4.99 |
48. | Jouets et ours  | Raggedy Ann         |       4.99 |
49. | Bear Emporium   | King doll           |       9.49 |
```

```
| Bears R Us      | King doll           |       9.49 |
| Doll House Inc. | King doll           |       9.49 |
| Fun and Games   | King doll           |       9.49 |
| Furball Inc.    | King doll           |       9.49 |
| Jouets et ours  | King doll           |       9.49 |
| Bear Emporium   | Queen doll          |       9.49 |
| Bears R Us      | Queen doll          |       9.49 |
| Doll House Inc. | Queen doll          |       9.49 |
| Fun and Games   | Queen doll          |       9.49 |
| Furball Inc.    | Queen doll          |       9.49 |
| Jouets et ours  | Queen doll          |       9.49 |
+-----------------+---------------------+------------+
54 rows in set (0.00 sec
```

上面的例子包含很多，不正确的数据。

内部联结

基于两边直接的相对测试，称为等值联结(euqijoin)

```
mysql> SELECT vend_name, prod_name, prod_price
    -> FROM Vendors INNER JOIN Products
    -> ON Vendors.vend_id = Products.vend_id;
+-----------------+---------------------+------------+
| vend_name       | prod_name           | prod_price |
+-----------------+---------------------+------------+
| Bears R Us      | 8 inch teddy bear   |       5.99 |
| Bears R Us      | 12 inch teddy bear  |       8.99 |
| Bears R Us      | 18 inch teddy bear  |      11.99 |
| Doll House Inc. | Fish bean bag toy   |       3.49 |
| Doll House Inc. | Bird bean bag toy   |       3.49 |
| Doll House Inc. | Rabbit bean bag toy |       3.49 |
| Doll House Inc. | Raggedy Ann         |       4.99 |
| Fun and Games   | King doll           |       9.49 |
| Fun and Games   | Queen doll          |       9.49 |
+-----------------+---------------------+------------+
9 rows in set (0.01 sec)
```

联结多个表

先列出所有列，再定义表之间的关系。

```
mysql> SELECT prod_name, vend_name, prod_price, quantity
    -> FROM OrderItems, Products, Vendors
    -> WHERE Products.vend_id = Vendors.vend_id
    -> AND OrderItems.prod_id = Products.prod_id
    -> AND order_num = 20007;
+--------------------+----------------+------------+----------+
| prod_name          | vend_name      | prod_price | quantity |
+--------------------+----------------+------------+----------+
| 18 inch teddy bear | Bears R Us     |      11.99 |       50 |
| Fish bean bag toy  | Doll House Inc. |       3.49 |      100 |
| Bird bean bag toy  | Doll House Inc. |       3.49 |      100 |
| Rabbit bean bag toy | Doll House Inc. |       3.49 |      100 |
| Raggedy Ann        | Doll House Inc. |       4.99 |       50 |
+--------------------+----------------+------------+----------+
5 rows in set (0.00 sec)
```

返回订购产品RGAN01的客户列表

```
mysql> SELECT cust_name, cust_contact
    -> FROM Customers
    -> WHERE cust_id IN (SELECT cust_id
    -> FROM Orders
    -> WHERE order_num IN (SELECT order_num
    -> FROM OrderItems
    -> WHERE prod_id = 'RGAN01'));
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
2 rows in set (0.00 sec)
```

下面使用联结查询

```
mysql> SELECT cust_name, cust_contact
    -> FROM Customers, Orders, OrderItems
    -> WHERE Customers.cust_id = Orders.cust_id
    -> AND OrderItems.order_num = Orders.order_num
    -> AND prod_id = 'RGAN01';
+---------------+--------------------+
```

```
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
2 rows in set (0.00 sec)
```

解释:返回的数据需要使用3个表，三个WHERE子句，最后过滤出RGAN01产品的数据

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter11.html*

12. 创建高级链接

创建高级链接

使用表别名

- 目的在于缩短SQL语句
- 运行单条SELECT 语句中多次使用相同的表。

```
mysql> SELECT cust_name, cust_contact FROM Customers AS C, Orders AS O,
OrderItems AS OI WHERE C.cust_id = O.cust_id AND OI.order_num =
O.order_num AND prod_id = 'RGAN01';
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
2 rows in set (0.01 sec)
```

解释：FROM中的子句有3个表，分别设置别名 `Customers AS C` ， `Orders AS O` ， `OrderItems AS OI` 给予子句WHERE引用。

自联结

```
mysql> SELECT cust_id, cust_name, cust_contact
    -> FROM Customers
    -> WHERE cust_name = (SELECT cust_name
    -> FROM Customers
    -> WHERE cust_contact = 'Jim Jones');
+------------+-----------+--------------------+
| cust_id    | cust_name | cust_contact       |
+------------+-----------+--------------------+
| 1000000003 | Fun4All   | Jim Jones          |
| 1000000004 | Fun4All   | Denise L. Stephens |
+------------+-----------+--------------------+
2 rows in set (0.01 sec)
```

解释；括号里的SELECT做了一个简单的检索，返回公司的cust_name，给予括号外SELECT查询。

另一个种查询方式

```
mysql> SELECT c1.cust_id, c1.cust_name, c1.cust_contact
    -> FROM Customers AS c1, Customers AS c2
    -> WHERE c1.cust_name = c2.cust_name
    -> AND c2.cust_contact = 'Jim Jones';
+------------+-----------+-------------------+
| cust_id    | cust_name | cust_contact      |
+------------+-----------+-------------------+
| 1000000003 | Fun4All   | Jim Jones         |
| 1000000004 | Fun4All   | Denise L. Stephens |
+------------+-----------+-------------------+
2 rows in set (0.01 sec)
```

自然联结

通过对表使用通配符*，对所有其他的表列，使用明确的子集来完成。

```

mysql> SELECT C.*, O.order_num, O.order_date, OI.prod_id,
    -> OI.quantity, OI.item_price
    -> FROM Customers AS C, Orders AS O, OrderItems AS OI
    -> WHERE C.cust_id = O.cust_id
    -> AND OI.order_num = O.order_num
    -> AND prod_id = 'RGAN01';
+------------+--------------+--------------------+-----------+------------+--
--------+-------------+-------------------+-----------------------+----------
-+--------------------+---------+----------+------------+
| cust_id    | cust_name     | cust_address        | cust_city | cust_state |
cust_zip | cust_country | cust_contact       | cust_email            |
order_num | order_date          | prod_id | quantity | item_price |
+------------+--------------+--------------------+-----------+------------+--
--------+-------------+-------------------+-----------------------+----------
-+--------------------+---------+----------+------------+
| 1000000004 | Fun4All       | 829 Riverside Drive | Phoenix    | AZ         |
88888    | USA          | Denise L. Stephens | dstephens@fun4all.com |
20007 | 2004-01-30 00:00:00 | RGAN01  |       50 |       4.49 |
| 1000000005 | The Toy Store | 4545 53rd Street    | Chicago    | IL         |
54545    | USA          | Kim Howard         | NULL                  |
20008 | 2004-02-03 00:00:00 | RGAN01  |        5 |       4.99 |
+------------+--------------+--------------------+-----------+------------+--
```

```
    --------+-------------+--------------------+-----------------------+----------
    -+---------------------+---------+----------+------------+
14. 2 rows in set (0.01 sec)
```

解释：*通配符只对第一个表使用，列出其他明确的列。

外部联结

联结包含那些在相关表中没有关联的行的行。

- 对每个客户下了多少订单进行计算，包括未下单的客户
- 列出所有产品以及订购数量，包含没有人订购的产品
- 计算平均销售规模，包括没下单的客户。
  检索所有客户及订单，内部联结

```
1.  mysql> SELECT Customers.cust_id, Orders.order_num
2.      -> FROM Customers INNER JOIN Orders
3.      -> ON Customers.cust_id = Orders.cust_id;
4.  +------------+-----------+
5.  | cust_id    | order_num |
6.  +------------+-----------+
7.  | 1000000001 |     20005 |
8.  | 1000000001 |     20009 |
9.  | 1000000003 |     20006 |
10. | 1000000004 |     20007 |
11. | 1000000005 |     20008 |
12. +------------+-----------+
13. 5 rows in set (0.00 sec)
```

外部联结，检索所有客户，包含那些没有订单的客户。

```
1.  mysql> SELECT Customers.cust_id, Orders.order_num
2.      -> FROM Customers LEFT OUTER JOIN Orders
3.      -> ON Customers.cust_id = Orders.cust_id;
4.  +------------+-----------+
5.  | cust_id    | order_num |
6.  +------------+-----------+
7.  | 1000000001 |     20005 |
8.  | 1000000001 |     20009 |
9.  | 1000000002 |      NULL |
10. | 1000000003 |     20006 |
11. | 1000000004 |     20007 |
```

```
| 1000000005 |     20008 |
+------------+-----------+
6 rows in set (0.00 sec)
```

OUTER JOIN 指定联结类型，与内部联结关联两个表中不同的是，外部联结还包含没有关联的行，用RIGHT与LEFT关键字指定包含其所有行的表是左边还是右边。

使用带聚集函数的联结

检索所有客户及每个客户所下订单

```
mysql> SELECT Customers.cust_id, COUNT(Orders.order_num) AS num_ord
    -> FROM Customers INNER JOIN Orders
    -> ON Customers.cust_id = Orders.cust_id
    -> GROUP BY Customers.cust_id;
+------------+---------+
| cust_id    | num_ord |
+------------+---------+
| 1000000001 |       2 |
| 1000000003 |       1 |
| 1000000004 |       1 |
| 1000000005 |       1 |
+------------+---------+
4 rows in set (0.02 sec)
```

解释： `INNER JOIN` 联结 `Customers` 跟 `Orders` 表，GROUP BY子句按客户分组数据，`COUNT(Orders.order_num)` 计算客户订单计数。

使用联结条件

- 主要联结类型，一般使用内部联结
- 不同的DBMS联结方式不同。
- 保证使用正确的联结条件
- 使用多个联结，先分别测试每个联结。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter12.html*

13. 组合查询

组合查询

如何用UNION操作符将多条SELECT语句组合成一个查询

组合查询

- 并(union) 执行多个查询并将结果作为单个查询结果返回。
  一般需要使用组合查询的情况
- 单个查询中从不同的表类似返回结果数据
- 单个表执行多个查询，按单个查询返回数据

创建组合查询

检索 IL,IN,MI几个洲的客户报表。

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_state IN ('IL', 'IN', 'MI');
+---------------+--------------+-----------------------+
| cust_name     | cust_contact | cust_email            |
+---------------+--------------+-----------------------+
| Village Toys  | John Smith   | sales@villagetoys.com |
| Fun4All       | Jim Jones    | jjones@fun4all.com    |
| The Toy Store | Kim Howard   | NULL                  |
+---------------+--------------+-----------------------+
3 rows in set (0.01 sec)
```

SELECT利用=符合，检索出所有Fun4All单位

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_name = 'Fun4All';
+-----------+--------------------+-----------------------+
| cust_name | cust_contact       | cust_email            |
+-----------+--------------------+-----------------------+
| Fun4All   | Jim Jones          | jjones@fun4all.com    |
```

```
| Fun4All    | Denise L. Stephens | dstephens@fun4all.com |
+-----------+--------------------+-----------------------+
2 rows in set (0.00 sec)
```

把上面两条语句进行组合

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_state IN ('IL','IN','MI')
    -> UNION
    -> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_name = 'Fun4All';
+---------------+--------------------+-----------------------+
| cust_name     | cust_contact       | cust_email            |
+---------------+--------------------+-----------------------+
| Village Toys  | John Smith         | sales@villagetoys.com |
| Fun4All       | Jim Jones          | jjones@fun4all.com    |
| The Toy Store | Kim Howard         | NULL                  |
| Fun4All       | Denise L. Stephens | dstephens@fun4all.com |
+---------------+--------------------+-----------------------+
4 rows in set (0.02 sec)
```

解释：中介有UNION分割开，并把输出组合成儿一个查询结果

另一个检索

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_state IN('IL','IN','MI')
    -> OR cust_name = 'Fun4All';
+---------------+--------------------+-----------------------+
| cust_name     | cust_contact       | cust_email            |
+---------------+--------------------+-----------------------+
| Village Toys  | John Smith         | sales@villagetoys.com |
| Fun4All       | Jim Jones          | jjones@fun4all.com    |
| Fun4All       | Denise L. Stephens | dstephens@fun4all.com |
| The Toy Store | Kim Howard         | NULL                  |
+---------------+--------------------+-----------------------+
4 rows in set (0.00 sec)
```

使用UNION规则

- 必须有两条以上SELECT语句组合，语句直接用关键字UNION分割。
- UNION中每个查询必须包含相同的列，表单式，聚集函数。
- 列的数据必须兼容，

是否带有重复行

UNION默认去掉重复行

如果想要所有行，可以使用UNION ALL 而不是UNION。

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_state IN ('IL','IN','MI')
    -> UNION ALL
    -> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_name = 'Fun4All';
+---------------+--------------------+-----------------------+
| cust_name     | cust_contact       | cust_email            |
+---------------+--------------------+-----------------------+
| Village Toys  | John Smith         | sales@villagetoys.com |
| Fun4All       | Jim Jones          | jjones@fun4all.com    |
| The Toy Store | Kim Howard         | NULL                  |
| Fun4All       | Jim Jones          | jjones@fun4all.com    |
| Fun4All       | Denise L. Stephens | dstephens@fun4all.com |
+---------------+--------------------+-----------------------+
5 rows in set (0.00 sec)
```

与上面例子比多了一行。

对组合查询结果排序

```
mysql> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_state IN ('IL''IN','MI')
    -> UNION
    -> SELECT cust_name, cust_contact, cust_email
    -> FROM Customers
    -> WHERE cust_name = 'Fun4ALL'
    -> ORDER BY cust_name, cust_contact;
```

```
+--------------+--------------------+------------------------+
| cust_name    | cust_contact       | cust_email             |
+--------------+--------------------+------------------------+
| Fun4All      | Denise L. Stephens | dstephens@fun4all.com  |
| Fun4All      | Jim Jones          | jjones@fun4all.com     |
| Village Toys | John Smith         | sales@villagetoys.com  |
+--------------+--------------------+------------------------+
3 rows in set (0.00 sec)
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter13.html*

14. 插入数据库

插入数据库

利用INSERT语句将数据插入表中

数据插入

用来插入(添加)行到数据库。

- 插入完整的行
- 插入行的一部分
- 插入某些查询结果

插入完整的行

指定表名和被插入到新行中的值

编写依赖与特定列次序的SQL语句，这样做有时会出错，但编写方便。

```
mysql> INSERT INTO Customers
    -> VALUES('1000000006',
    -> 'Toy Land',
    -> '123 Any Street',
    -> 'New York',
    -> 'NY',
    -> '11111',
    -> 'USA',
    -> NULL,
    -> NULL);
Query OK, 1 row affected (0.01 sec)
```

解释：插入一个新客户到Customers表，存储到每个表列的数据VALUES子句中给出，对每个表必须提供一个值，如果某列没值，就应该使用NULL值。

养成指定顺序插入数据，虽然写起来繁琐，但不容易发生错误。注意每个列，都必须提供一个值。

```
mysql> INSERT INTO Customers(cust_id,
    -> cust_contact,
    -> cust_email,
    -> cust_name,
```

```
    -> cust_address,
    -> cust_city,
    -> cust_state,
    -> cust_zip)
    -> VALUES('1000000009',
    -> NULL,
    -> NULL,
    -> 'Toy Land',
    -> '123 Any Street',
    -> 'New York',
    -> 'NY',
    -> '11111');
```

插入部分行

指定某列提供值，其他的不提供值

```
mysql> INSERT INTO Customers(cust_id,
    -> cust_name,
    -> cust_address,
    -> cust_city,
    -> cust_state,
    -> cust_zip,
    -> cust_country)
    -> VALUES('1000000008',
    -> 'Toy Land',
    -> '123 Any Street',
    -> 'New York',
    -> 'NY',
    -> '11111',
    -> 'USA');
Query OK, 1 row affected (0.01 sec)
```

解释:忽略表中cust_ontact 与cust_email 值。

插入部分值，前提条件是表允许

- 改列定义为允许NULL值
- 表改成默认值，如果不给，将使用默认
  如果没有这两个前天条件，就服务插入部分值。

插入检索出的数据

利用SELECT 语句的输出结果插入表中，INSERT SELECT两条结合。

新键一个表

```
mysql> CREATE TABLE CustomersNew
    -> (
    ->   cust_id      char(10)  NOT NULL ,
    ->   cust_name    char(50)  NOT NULL ,
    ->   cust_address char(50)  NULL ,
    ->   cust_city    char(50)  NULL ,
    ->   cust_state   char(5)   NULL ,
    ->   cust_zip     char(10)  NULL ,
    ->   cust_country char(50)  NULL ,
    ->   cust_contact char(50)  NULL ,
    ->   cust_email   char(255) NULL
    -> );
Query OK, 0 rows affected (0.06 sec)
```

插入是注意主键值不能重复。

```
mysql> INSERT INTO CustomersNew(cust_id,
    -> cust_contact,
    -> cust_email,
    -> cust_name,
    -> cust_address,
    -> cust_city,
    -> cust_state,
    -> cust_zip,
    -> cust_country)
    -> SELECT cust_id,
    -> cust_contact,
    -> cust_email,
    -> cust_name,
    -> cust_address,
    -> cust_city,
    -> cust_state,
    -> cust_zip,
    -> cust_country
    -> FROM Customers;
```

解释:将 `CustomersNew` 所有的数据导入 `Customers`

从一个表复制到另一个表

只复制表结构，不复制数据。

```
mysql> CREATE TABLE Customers_New like Customers;
Query OK, 0 rows affected (0.03 sec)
```

创建一个CustCopy 表，并把Customers表中的数据复制过来。

```
mysql> CREATE TABLE CustCopy AS
    -> SELECT *
    -> FROM Customers;
Query OK, 8 rows affected (0.03 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

查看一个表是如果创建的

```
mysql> SHOW CREATE TABLE Customers;
```

查看表的结构

```
mysql> DESC Customers
```

注意：

- 任何SELECT 选择的子句都可以使用。WHERE ,GROUP BY等
- 可以利用链接从多个表插入数据
- 不过从多少个表检索出来的数据，数据都只能插入一个表中。
  SELECT INTO 可以用来测试SQL语句前，复制一个表出来测试，避免影响原来的表。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter14.html*

15. 更新和删除数据

更新和删除数据

利用UPDATE和DELETE语句进行操作表数据。

更新数据

UPDATE用来更新修改表中的数据

- 更新表中特定的行
- 更新表中所有行
  注意： 如果省略了WHERE子句，`就会更新所有行。`

UPDATE语句有三个部分组合

- 要更新的表
- 列名和他们的新值
- 确定要更新哪些行的过滤条件

```
mysql> UPDATE Customers
   -> SET cust_email = 'Kim@gmail.com'
   -> WHERE cust_id = '1000000005';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

解释： SET命令用来建新值赋予给更新的列，设置了cust_email列为指定的值。`SET cust_email = 'Kim@gmail.com'` WHERE子句告诉要更新哪一行，如果没有WHERE子句，电子邮件将会更新Customers表中所有的行。

更新一列多个值

```
mysql> UPDATE Customers
    -> SET cust_contact = 'Sam Roberts',
    -> cust_email = 'sam@toyland.com'
    -> WHERE cust_id = '1000000006';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

解释： 使用SET命令，每个‘列=值’用逗号隔开，区分多个列。

更新某个列NULL

可以把列设置成NULL，如果表允许设置NULL

```
mysql> UPDATE Customers
    -> SET cust_email = NULL
    -> WHERE cust_id = '1000000005';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

删除数据

使用 `DELETE` 从数据库删除数据，

- 从表中删除特定的行
- 从表中删除所有的行

```
mysql> DELETE FROM Customers WHERE cust_id = '1000000009';
Query OK, 1 row affected (0.01 sec)
```

解释:指定删除 表Customers 中的数据

注意:

- DELETE语句删除行，但不能删除表本身
- 想删除表中所有的行，可以使用TRUNCATE TABLE语句

更新和删除的原则

- UPDATE跟DELETE语句都具有WHERE子句，如果忽略WHERE子句建会应用到所有行，所以除非更新所有行
- 保证每个表都有主键，WHERE应用到主键。
- 使用UPDATE和DELETE语句前先，先SELECT进行测试，确保编写的WHERE子句正确。
- 使用强制实施引用完整性数据库，防止误删除行。
- 现在MYSQL不带有WHERE子句的UPDATE或DELETE子句执行。

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter15.html*

16. 创建和操作表

创建和操作表

创建表

创建表的方法

- 直接用交互创建表和管理工具
- 表也可以直接用SQL语句操纵

表创建基础

利用CREATE TABLE创建表，需以下信息

- 新表的名字，在关键字CREATE TABLE之后
- 表列的名字和定义，用逗号隔开。
- 部分DBMS需要指定表的位置。

```
mysql> CREATE TABLE Products
  -> ( prod_id CHAR(10) NOT NULL,
  -> vend_id CHAR(10) NOT NULL,
  -> prod_name CHAR(254) NOT NULL,
  -> prod_price DECIMAL(8,2) NOT NULL,
  -> prod_desc TEXT(1000) NULL
  -> );
```

解释：表名后面跟着的列，是表的定义，括在园括号之中，各列直接用逗号分割，一共有五个表，列名后面跟数据类型，圆括号结束。

使用NULL值

允许使用NULL值的列，也允许插入行时不给出该列的值，如果不允许该行没值，可以使用NULL

```
mysql> CREATE TABLE Order_s
    -> (
    -> order_num INTEGER NOT NULL,
    -> order_date DATETIME NOT NULL,
    -> cust_id CHAR(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.02 sec)
```

使用列的含义使用NOT NULL来插入，阻止插入的列没事值。

混合NULL与NOT NULL列的表

```
mysql> CREATE TABLE Vendors
    -> (
    -> vend_id CHAR(10) NOT NULL,
    -> vend_name CHAR(50) NOT NULL,
    -> vend_address CHAR(50),
    -> vend_city CHAR(50),
    -> vend_state CHAR(5),
    -> vend_zip CHAR(10),
    -> vend_country CHAR(50)
    -> );
```

指定默认值

插入行时，如果不指定值，自动采用默认值

```

mysql> CREATE TABLE OrderItems
    -> (
    -> order_num INTEGER NOT NULL,
    -> order_item INTEGER NOT NULL,
    -> prod_id CHAR(10) NOT NULL,
    -> quantity INTEGER NOT NULL DEFAULT 1,
    -> item_price DECIMAL(8,2) NOT NULL
    -> );
```

解释: quantity 用于存放订单中的物品，给了一个默认值 `DEFAULT 1` ，如果没有给值，默认是1

默认值经常用于日期跟时间戳

- Mysql 用户指定DEFAULT CURRENT_DATE()

更新表

使用 `ALTER TABLE` 语句来更新内容，但有几个地方要注意

- 表中包含数据是不用对其进行更新。
- 所有的DBMS都允许表添加列
- 许多DBMS不允许删除更新列
- ALTER TABLE 之后给出要更改的表面。

- 所做更改的列表

```
mysql> ALTER TABLE Vendors
  -> ADD vend_phone CHAR(20);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

解释:给Vendors 表添加一个列，数据类型为CHAR

删除列

```
mysql> ALTER TABLE Vendors
    -> DROP COLUMN vend_phone;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

注意: 使用ALTER TABLE 过程是无法撤销的，所以操作之前，需要进行备份。

删除表

使用DROP TABLE来删除整个表

```
mysql> DROP TABLE CustCopy;
Query OK, 0 rows affected (0.02 sec)
```

删除整个 `CustCopy` 表。DROPY无法删除带有关系规则的表。

重命名表

使用RENAME语句来重命名表

```
mysql> RENAME TABLE Vendor TO Vendors;
Query OK, 0 rows affected (0.01 sec)
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter16.html*

17. 使用视图

使用视图

如果利用视图来简化执行SQL语句某些操作。

视图

视图是虚拟表，只是在使用时动态检索数据查询。

为什么使用视图函数

- 重用SQL语句
- 简化复杂的SQL操作，方便重用
- 使用表的组成部分而不是全部
- 包含数据，可以给用于提高表的特定访问权限，而不是整个。
- 更改数据格式和表示，当需要返回跟底层表的表示格式不同的数据。
  创建视图函数后，可以用与表相同的方式利用他们，可以进行SELECT执行操作，过滤，排序数据
  将视图结合其他的视图或表，甚至能添加和更新数据。

注意： 知道视图仅仅是用来查看跟存储在别处的数据，本身不包含数据，数据是从其他表检索出来，更改与添加表中的数据时，视图将返回改变的数据。

视图函数限制规则

- 表名必须唯一
- 创建视图函数没有限制数目
- 有足够的权限
- 可以利用其他视图中的数据，来查询构造一个新的视图。
- 视图不能索引，也不能关联默认值

创建视图

`CREATE VIEW` 语句来创建视图。删除视图用 `DROP VIEW viewname` ;

```
1.
2.  mysql> CREATE VIEW ProductCustomers AS
3.      -> SELECT cust_name, cust_contact, prod_id
4.      -> FROM Customers, Orders, OrderItems
5.      -> WHERE Customers.cust_id = Orders.cust_id
```

```
    -> AND OrderItems.order_num = Orders.order_num;
Query OK, 0 rows affected (0.03 sec)
```

创建一个ProductCustomers的视图，联结三个表，返回已经订购任意产品的客户

```
mysql> SELECT * FROM ProductCustomers;
+---------------+--------------------+---------+
| cust_name     | cust_contact       | prod_id |
+---------------+--------------------+---------+
| Village Toys  | John Smith         | BR01    |
| Village Toys  | John Smith         | BR03    |
| Village Toys  | John Smith         | BNBG01  |
| Village Toys  | John Smith         | BNBG02  |
| Village Toys  | John Smith         | BNBG03  |
| Fun4All       | Jim Jones          | BR01    |
| Fun4All       | Jim Jones          | BR02    |
| Fun4All       | Jim Jones          | BR03    |
| Fun4All       | Denise L. Stephens | BR03    |
| Fun4All       | Denise L. Stephens | BNBG01  |
| Fun4All       | Denise L. Stephens | BNBG02  |
| Fun4All       | Denise L. Stephens | BNBG03  |
| Fun4All       | Denise L. Stephens | RGAN01  |
| The Toy Store | Kim Howard         | RGAN01  |
| The Toy Store | Kim Howard         | BR03    |
| The Toy Store | Kim Howard         | BNBG01  |
| The Toy Store | Kim Howard         | BNBG02  |
| The Toy Store | Kim Howard         | BNBG03  |
+---------------+--------------------+---------+
18 rows in set (0.01 sec)
```

检索 ProductCustomers表的数据

```

mysql> SELECT cust_name, cust_contact
    -> FROM ProductCustomers
    -> WHERE prod_id = 'RGAN01';
+---------------+--------------------+
| cust_name     | cust_contact       |
+---------------+--------------------+
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |
+---------------+--------------------+
```

```
11. 2 rows in set (0.00 sec)
```

解释： 用WHERE子句过滤实体中检索的特定数据。

用视图重新格式化检索出的数据

假设经常需要检索下面的语句，为了不用经常执行，把此语句转换为视图。

```
1. mysql> SELECT CONCAT(vend_name, ' (', vend_country, ')') AS vend_title FROM
   Vendors
2.     -> ORDER BY vend_name;
3. +--------------------------+
4. | vend_title               |
5. +--------------------------+
6. | Bear Emporium (USA)      |
7. | Bears R Us (USA)         |
8. | Doll House Inc. (USA)    |
9. | Fun and Games (England)  |
10. | Furball Inc. (USA)      |
11. | Jouets et ours (France) |
12. +--------------------------+
13. 6 rows in set (0.00 sec)
```

把上面的语句转成视图

```
1.
2. mysql> CREATE VIEW VendorLocations AS SELECT CONCAT(vend_name, ' (',
   vend_country, ')') AS vend_title FROM Vendors ORDER BY vend_name;
3. Query OK, 0 rows affected (0.02 sec)
```

检索新生成的视图表

```
1. mysql> SELECT * FROM VendorLocations;
2. +--------------------------+
3. | vend_title               |
4. +--------------------------+
5. | Bear Emporium (USA)      |
6. | Bears R Us (USA)         |
7. | Doll House Inc. (USA)    |
8. | Fun and Games (England)  |
9. | Furball Inc. (USA)       |
10. | Jouets et ours (France) |
```

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

用视图过滤不想要的数据

定一个一个emaillist，需要过滤没有email的邮件地址的客户。

```
mysql> CREATE VIEW CustomerEMAIList AS
    -> SELECT cust_id, cust_name, cust_email
    -> FROM Customers
    -> WHERE cust_email IS NOT NULL;
Query OK, 0 rows affected (0.03 sec)
```

解释：要WHERE 子句过滤没有电子邮箱的客户。

```
mysql> SELECT * FROM CustomerEMAIList;
+------------+--------------+----------------------+
| cust_id    | cust_name    | cust_email           |
+------------+--------------+----------------------+
| 1000000001 | Village Toys | sales@villagetoys.com |
| 1000000003 | Fun4All      | jjones@fun4all.com    |
| 1000000004 | Fun4All      | dstephens@fun4all.com |
| 1000000006 | Toy Land     | sam@toyland.com       |
+------------+--------------+----------------------+
4 rows in set (0.01 sec)
```

使用视图计算字段

检索订单物品，计算价格

```
mysql> SELECT prod_id,
    -> quantity,
    -> item_price,
    -> quantity*item_price AS expaned_price
    -> FROM OrderItems
    -> WHERE order_num = 20008;
+---------+----------+------------+---------------+
| prod_id | quantity | item_price | expaned_price |
+---------+----------+------------+---------------+
| RGAN01  |        5 |       4.99 |         24.95 |
| BR03    |        5 |      11.99 |         59.95 |
```

```
| BNBG01  |       10 |       3.49 |         34.90 |
| BNBG02  |       10 |       3.49 |         34.90 |
| BNBG03  |       10 |       3.49 |         34.90 |
+---------+----------+------------+---------------+
5 rows in set (0.01 sec)
```

换成视图

```
mysql> CREATE VIEW OrderItemsExpandes AS
    -> SELECT order_num,
    -> prod_id,
    -> quantity,
    -> item_price,
    -> quantity*item_price AS expanded_price
    -> FROM OrderItems;
Query OK, 0 rows affected (0.01 sec)
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter17.html*

18. 管理事务处理

管理事务处理

事务处理(transaction processing)

可以用来维护数据的完整性，保证SQL的操作要么完全执行，要么完全不执行，如果发生错误就进行撤销。

- 保证数据的完整性。
- 保证数据不受外影响。
  事务处理的几道术语
- 事务(transaction) 一组SQL语句
- 退回(rollback)撤销执行SQL语句的过程
- 提交(commit) 将为执行的SQL语句写入数据库表
- 保留点(savepoint) 临时存储点，用于发布退回。
  事务操作简单的例子

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM Vendor_n;
Query OK, 6 rows affected (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> COMMIT; 提交操作。
```

设置保留点

```
mysql> SAVEPOINT delete_vendor;
Query OK, 0 rows affected (0.00 sec)

mysql> ROLLBACK TO delete_vendor;
Query OK, 0 rows affected (0.00 sec)
```

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/chapter19.html*

19. 备份脚本

MySQL 脚本备份相关

Shell 脚本定期备份

mysql_config_editor — MySQL Configuration Utility 需要配置下 `mysql_config_editor`

```
mysql_config_editor set --login-path=client
      --host=localhost --user=localuser --password
```

`backup_parent_dir` 是备份路径 `mysql_user="root"` 备份用户，其实这里可以不用

```
#!/bin/bash
# Simple script to backup MySQL databases

# Parent backup directory
backup_parent_dir="./backup"

# MySQL settings
mysql_user="root"

# Read MySQL password from stdin if empty

# Check MySQL password
#echo exit | mysql --login-path=client  --host=localhost --user=${mysql_user} --password=${mysql_password} -B 2>/dev/null
echo exit | mysql --login-path=client -B 2>/dev/null
if [ "$?" -gt 0 ]; then
  echo "MySQL ${mysql_user} password incorrect"
  exit 1
else
  echo "MySQL ${mysql_user} password correct."
fi

# Create backup directory and set permissions
backup_date=`date +%Y_%m_%d_%H_%M`
backup_dir="${backup_parent_dir}/${backup_date}"
echo "Backup directory: ${backup_dir}"
```

```
mkdir -p "${backup_dir}"
chmod 700 "${backup_dir}"
logfile="$backup_parent_dir/"backup_log_"$(date +'%Y_%m')".txt

# Get MySQL databases
mysql_databases=`echo 'show databases' | mysql --login-path=client | sed /^Database$/d`

# Backup and compress each database
for database in $mysql_databases
do
  if [ "${database}" == "information_schema" ] || [ "${database}" == "performance_schema" ]; then
        additional_mysqldump_params="--skip-lock-tables"
  else
        additional_mysqldump_params=""
  fi
  echo "Creating backup of \"${database}\" database"
  mysqldump --login-path=client  ${additional_mysqldump_params} -e  | gzip > "${backup_dir}/${database}.gz"
  echo "mysqldump finished at $(date +'%d-%m-%Y %H:%M:%S')" >> "$logfile"
  chmod 600 "${backup_dir}/${database}.gz"
done
echo "mysqldump finished at $(date +'%d-%m-%Y %H:%M:%S')" >> "$logfile"
```

再用crontab 来定制备份计划

python 备份脚本

```
import os
import time

filestamp = time.strftime("%Y-%m-%d")

database_list_command="mysql --login-path=client -e 'show databases'"
for database in os.popen(database_list_command).readlines():
    database = database.strip()
    if database == 'information_schema':
        continue
    if database == 'performance_schema':
        continue
    filename = '/Users/xiaodian/mysql/backup/{0}_{1}.sql'.format(database,
```

```
    filestamp)
14.     os.popen("mysqldump --login-path=client -c {0}| gzip >
    {1}.gz".format(database, filename))
15.
16. os.popen("echo mysqldump finished at {0} >>
    backup_log_{1}.txt".format(filename, filestamp))
```

脚本参考https://blog.sleeplessbeastie.eu/2012/11/22/simple-shell-script-to-backup-mysql-databases/

> 原文: *https://cxiaodian.gitbooks.io/mysql/content/script_back.html*