Chapter 1. 了解 SQL

1.1. 数据库基础

1.1.1. 什么是数据库

1、数据库是一个以某种有组织的方式存储的数据集合

- 理解数据库的一种最简单的办法是将其想象为一个文件柜
- 此文件柜是一个存放数据的物理位置，不管数据是什么以及如何组织的

2、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)**

3、误区

- 误用导致混淆人们通常用数据库这个术语来代表他们使用的数据库软件这是不正确的，它是引起混淆的根源
- 确切地说，数据库软件应称为 DBMS(数据库管理系统)
- **数据库是通过 DBMS 创建和操纵的容器**
- 数据库可以是保存在硬设备上的文件，但也可以不是
- 在很大程度上说，数据库究竟是文件还是别的什么东西并不重要，因为你并不直接访问数据库;你使用的是 DBMS，它替你访问数据库

1.1.2. 表

1、表(table)：**某种特定类型数据的结构化清单**

2、数据库中的每个表都有一个名字，用来标识自己。此名字是唯一的，这表示数据库中没有其他表具有相同的名字

3、模式(schema)：**关于数据库和表的布局及特性的信息**

- 有时，模式用作数据库的同义词
- 遗憾的是，模式的含义通常在上下文中并不是很清晰
- 本书中，模式指的是上面给出的定义

1.1.3. 列和数据类型

1、表由列组成。列中存储着表中某部分的信息

2、列(column)：**表中的一个字段**。所有表都是由一个或多个列组成的

3、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

1.1.4. 行

1、行(row)：**表中的一个记录**

1.1.5. 主键

1、主键(primary key)：**一列(或一组列)，其值能够唯一区分表中每个行**

2、唯一标识表中每行的这个列(或这组列)称为主键。主键用来表示一个特定的行。没有主键，更新或删除表中特定行很困难，因为没有安全的方法保证只涉及相关的行

3、表中的任何列都可以作为主键，只要它满足以下条件

- 任意两行都不具有相同的主键值
- 每个行都必须具有一个主键值(主键列不允许 NULL 值)

4、主键的最好习惯：

- 不更新主键列中的值
- 不重用主键列的值
- 不在主键列中使用可能会更改的值(例如，如果使用一个名字作为主键以标识某个供应商，当该供应商合并和更改其名字时，必须更改这个主键)

1. 2. 什么是数据库

1、SQL(发音为字母 S-Q-L 或 sequel)是结构化查询语言(Structured Query Language)的缩写。**SQL 是一种专门用来与数据库通信的语言**

2、SQL 的优点

- SQL 不是某个特定数据库供应商专有的语言。几乎所有重要的 DBMS 都支持 SQL，所以，学习此语言使你几乎能与所有数据库打交道
- SQL 简单易学。它的语句全都是由描述性很强的英语单词组成，而且这些单词的数目不多
- SQL 尽管看上去很简单，但它实际上是一种强有力的语言，灵活使用其语言元素，可以进行非常复杂和高级的数据库操作

Chapter 2. MySQL 简介

2.1. 什么是 MySQL

1、数据的所有存储、检索、管理和处理实际上是由数据库软件——DBMS(数据库管理系统)完成的。**MySQL 是一种 DBMS，即它是一种数据库软件**

2、选择 MySQL 的原因

- 成本：MySQL 是开放源代码的，一般可以免费使用(甚至可以免费修改)。
- 性能：MySQL 执行很快(非常快)。
- 可信赖：某些非常重要和声望很高的公司、站点使用 MySQL，这些公司和站点都用 MySQL 来处理自己的重要数据
- 简单：MySQL 很容易安装和使用

3、MySQL 受到的唯一真正的批评是它并不总是支持其他 DBMS 提供的功能和特性。然而，这一点也正在逐步得到改善，MySQL 的各个新版本正不断增加新特性、新功能

2.1.1. 客户机---服务器软件

1、DBMS 可分为两类：

- 一类为：基于共享文件系统的 DBMS
- 另一类为：基于客户机—服务器的 DBMS
- 前者(包括诸如 Microsoft Access 和 FileMaker)用于桌面用途，通常不用于高端或更关键的应用

2、MySQL、Oracle 以及 Microsoft SQL Server 等数据库是基于客户机—服务器的数据库

- 客户机—服务器应用分为两个不同的部分
  - 服务器：负责所有数据访问和处理的一个软件。这个软件运行在称为数据库服务器的计算机上
  - 与数据文件打交道的只有服务器软件，关于数据、数据添加、删除 和数据更新的所有请求都由服务器软件完成
  - 这些请求或更改来自运行客户机软件的计算机
  - 客户机：与用户打交道的软件

2.1.2. MySQL 版本

1、<未完成>

2.2. MySQL 工具

1、<未完成>

2.2.1. mysql 命令行实用程序

2.2.2. MySQL Administrator

2.2.3. MySQL Query Browser

Chapter 3. 使用 MySQL

3.1. 连接

1、为了连接到 MySQL，需要以下信息

- 主机名(计算机名)：如果连接到本地 MySQL 服务器，为 localhost
- 端口：(如果使用默认端口 3306 之外的端口)
- 一个合法的用户名
- 用户口令(如果需要)

3.2. 选择数据库

1、可使用 USE 关键字选择一个数据库

3.3. 了解数据库和表

1、SHOW

- SHOW DATABASES; <==显示所有数据库
- SHOW TABLES; <==显示当前数据库中所有表
- SHOW COLUMNS FROM [表名]; <==该表中所有列的信息
- DESCRIBE [表
-]; <==同上
- SHOW STATUS; <==用于显示广泛的服务器状态
- SHOW CREATE DATABASE [数据库名]; <==显示创建特定数据库的 MySQL 语句
- SHOW CREATE TABLE [表明]; <==显示创建特定表的 MySQL 语句
- SHOW GRANTS：显示授予用户的安全权限
- SHOW ERRORS 和 SHOW WARNINGS <==用来显示服务器错误或警告消息

Chapter 4. 检索数据

4.1. SELECT 语句

1、为了使用 SELECT 检索表数据，必须至少给出两条信息

- 想选择什么
- 从什么地方选择

4.2. 检索单个列

1、SELECT [列名] FROM [表
]

2、多条 SQL 语句必须以分号(;)分隔

- 如同多数 DBMS 一样，MySQL 不需要在单条 SQL 语句后加分号
- 如果愿意总是可以加上分号，因为没有坏处
- **SQL 语句不区分大小写，SELECT 与 select 是相同的，但是习惯对 SQL 所有关键字使用大写，对列和表**
- **使用小写，这样做使代码更易于阅读和调试**
- 在处理 SQL 语句时，其中所有空格都被忽略。SQL 语句可以在一行上给出，也可以分成许多行。多数 SQL 开发人员认为将 SQL 语句分成多行更容易阅读和调试

4.3. 检索多个列

1、SELECT [列名 1,列名 2,列名 3,...,列名 n] FROM [表
]

2、SELECT 关键字后给出多个列名，列名之间必须以逗号分隔

- 在选择多个列时，一定要在列名之间加上逗号
- **但最后一个列名后不加**

4.4. 检索所有列

1、SELECT * FROM [表
]

2、一般，除非你确实需要表中的每个列，否则最好别使用*通配符。虽然使用通配符可能会使你自己省事，不用明确列出所需列，但检索不需要的列通常会降低检索和应用程序的性能

3、使用通配符有一个大优点。由于不明确指定列名(因为星号检索每个列)，所以能检索出名字未知的列

4.5. 检索不同的行

1、DISTINCT 关键字：顾名思义，此关键字指示 MySQL 只返回不同的值。

- **DISTINCT 关键字应用于所有列而不仅是前置它的列**

2、例子

```
SELECT DISTINCT vend_id
FROM products;
```

4.6. 限制结果

1、为了返回第一行或前几行，可使用 LIMIT 子句

2、例子 1

```
SELECT prod_name
FROM products
LIMIT 5;
```

3、例子 2

```
SELECT prod_name
FROM products
LIMIT 5,5;
```

- 第一个数为开始位置，第二个数为要检索的行数

4、几个注意点：

- 行 0：检索出来的第一行为行 0 而不是行 1。因此，LIMIT1,1 将检索出第二行而不是第一行
- 在行数不够 LIMIT 中指定要检索的行数为检索的最大行数。如果没有足够的行(例如，给出 LIMIT 10, 5，但只有 13 行)，MySQL 将只返回它能返回的那么多行
- MySQL 5 支持 LIMIT 的另一种替代语法。LIMIT 4 OFFSET 3 意为从行 3 开始取 4 行，就像 LIMIT 3, 4 一样

4. 7. 使用完全限定的表

4. 8.

1、同时使用表名和列名

```
SELECT products.prod_name
FROM products;
```

Chapter 5. 排序检索数据

5.1. 排序数据

1、关系数据库设计理论认为，如果不明确规定排序顺序，则不应该假定检索出的数据的顺序有意义

2、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

3、为了明确地排序用 SELECT 语句检索出的数据，可使用 ORDER BY 子句。ORDER BY 子句取一个或多个列的名字，据此对输出进行排序

4、例子

```
SELECT prod_name
FROM products
ORDER BY prod_name;
```

5、**通过非选择列进行排序**：通常，ORDERBY 子句中使用的列将是为显示所选择的列。但是，实际上并不一定要这样，**用非检索的列排序数据是完全合法的**

```
SELECT name
FROM students
ORDER BY age;
```

5.2. 按多个列排序

1、为了按多个列排序，只要指定列名，列名之间用逗号分开即可(就像选择多个列时所做的那样)

2、重要的是理解在按多个列排序时，排序完全按所规定的顺序进行

- 下一个被指定的列名仅会排列在上一次排列中顺序相同的行

5.3. 指定排序方向

1、数据排序不限于升序排序(从 A 到 Z)。这只是默认的排序顺序，还可以使用 ORDER BY 子句以降序(从 Z 到 A)顺序排序。为了进行降序排序，必须指定 DESC 关键字

2、例子

```
SELECT prod_id,prod_price,prod_name
FROM products
ORDER BY prod_price DESC, prod_name;
```

3、**DESC 关键字只应用到直接位于其前面的列名**

- 如果想在多个列上进行降序排序，必须对每个列指定 DESC 关键字

4、与 DESC 相反的关键字是 ASC(ASCENDING)，在升序排序时可以指定它。 但实际上，ASC 没有多大用处，因为升序是默认的(如果既不指定 ASC 也不指定 DESC，则假定为 ASC)。

5、区分大小写和排序顺序

- 在对文本性的数据进行排序时，A 与 a 相同吗?a 位于 B 之前还是位于 Z 之后?这些问题不是理论问题，其答案取决于数据库如何设置
- 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。但是，许多数据库管理员能够在需要时改变这种行为(如果你的数据库包含大量外语字符，可能必须这样做)

- 如果确实需要改变这种排序顺序，用简单的 ORDER BY 子句做不到。你必须请求数据库管理员的帮助

6、使用 ORDER BY 和 LIMIT 的组合，能够找出一个列中最高或最低的值

```
SELECT prod_price
FROM products
ORDER BY prod_price DESC
LIMIT 1;
```

- ORDERBY 子句的位置：在给出 ORDERBY 子句时，应该保证它位于 FROM 子句之后。如果使用 LIMIT，它必须位于 ORDER BY 之后
- 使用子句的次序不对将产生错误消息。

Chapter 6. 过滤数据

6.1. 使用 WHERE 子句

1、数据库表一般包含大量的数据，很少需要检索表中所有行。通常只会根据特定操作或报告的需要提取表数据的子集。只检索所需数据需要指定搜索条件(search criteria)，搜索条件也称为过滤条件(filter condition)

2、在 SELECT 语句中，数据根据 WHERE 子句中指定的搜索条件进行过滤。WHERE 子句在表名(FROM 子句)之后给出

```
SELECT prod_name,prod_price
FROM products
WHERE prod_price=2.50;
```

3、**SQL 过滤与应用过滤**：

- 数据也可以在应用层过滤。为此目的，SQL 的 SELECT 语句为客户机应用检索出超过实际所需的数据，然后客户机代码对返回数据进行循环，以提取出需要的行
- 通常，这种实现并不令人满意。因此，对数据库进行了优化，以便快速有效地对数据进行过滤。让客户机应用(或开发语言)处理数据库的工作将会极大地影响应用的性能，并且使所创建的应用完全不具备可伸缩性。此外，如果在客户机上过滤数据，服务器不得不通过网络发送多余的数据，这将导致网络带宽的浪费

4、WHERE 子句的位置：

- 在同时使用 ORDERBY 和 WHERE 子句时，应该让 ORDER BY 位于 WHERE 之后，否则将会产生错误

6.2. WHERE 子句操作符

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

6.2.1. 检查单个值

1、例子

```
SELECT name,age
FROM students
WHERE age<20;
```

6.2.2. 不匹配检查

1、例子

```
SELECT name,age
```

```
FROM students
WHERE age<>20;

SELECT name,age
FROM students
WHERE age!=20;
```

2、何时使用引号

- 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

6.2.3. 范围值检查

1、为了检查某个范围的值，可使用 BETWEEN 操作符。其语法与其他 WHERE 子句的操作符稍有不同，因为它需要两个值，即范围的开始值和结束值

```
SELECT name,age
FROM students
WHERE age BETWEEN 12 AND 30;
```

6.2.4. 空值检查

1、在创建表时，表设计人员可以指定其中的列是否可以不包含值。在一个列不包含值时，称其为包含空值 NULL

2、NULL：无值(no value)，**它与字段包含 0、空字符串或仅仅包含空格不同**

3、SELECT 语句有一个特殊的 WHERE 子句，可用来检查具有 NULL 值的列。这个 WHERE 子句就是 IS NULL 子句

```
SELECT age,tel
FROM products
WHERE name IS NULL;
```

4、NULL 与不匹配：

- 在通过过滤选择出不具有特定值的行时，你**可能希望**返回具有 NULL 值的行
- 但是，不行。**因为(NULL)未知具有特殊的含义，数据库不知道它们是否匹配，所以在匹配过滤或不匹配过滤时不返回它们**
- 因此，在过滤数据时，一定要验证返回数据中确实给出了被过滤列具有 NULL 的行。

Chapter 7. 数据过滤

7.1. 组合 WHERE 子句

1、MySQL 允许给出多个 WHERE 子句。这些子句可以两种方式使用：以 AND 子句的方式或 OR 子句的方式使用

2、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)

7.1.1. AND 操作符

1、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

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id=1003 AND prod_price<=10;
```

7.1.2. OR 操作符

1、OR 操作符与 AND 操作符不同，它指示 MySQL 检索匹配任一条件的行

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id=1002 OR vend_id=1003;
```

7.1.3. 计算次序

1、WHERE 可包含任意数目的 AND 和 OR 操作符。允许两者结合以进行复杂和高级的过滤

2、AND 的优先级高于 OR 的优先级，与其他语言类似

3、在 WHERE 子句中使用圆括号

- 任何时候使用具有 AND 和 OR 操作符的 WHERE 子句都应该使用圆括号明确地分组操作符
- 不要过分依赖默认计算次序，即使它确实是你想要的东西也是如此
- 使用圆括号没有什么坏处，它能消除歧义

```
SELECT prod_name, prod_price
FROM products
WHERE (vend_id=1002 OR vend_id=1003);
```

7.2. IN 操作符

1、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。IN 操作符后跟由逗号分隔的合法值清单，整个清单必须括在圆括号

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id IN (1002,1003)
ORDER BY prod_name;
```

2、IN 操作符可用 OR 操作符等价表示

3、IN 操作符的优势

- 在使用长的合法选项清单时，IN 操作符的语法更清楚且更直观
- 在使用 IN 时，计算的次序更容易管理(因为使用的操作符更少)
- IN 操作符一般比 OR 操作符清单执行更快

- IN 的最大优点是可以包含其他 SELECT 语句，使得能够更动态地建立 WHERE 子句

7.2.1. NOT 操作符

1、WHERE 子句中的 NOT 操作符有且只有一个功能，**那就是否定它之后所跟的任何条件**

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id NOT IN (1002,1003)
ORDER BY prod_name;
```

2、MySQL 中的 NOT：

- MySQL 支持使用 NOT 对 IN、BETWEEN 和 EXISTS 子句取反
- 这与多数其他 DBMS 允许使用 NOT 对各种条件取反有很大的差别

Chapter 8. 用通配符进行过滤

8.1. LIKE 操作符

1、通配符(wildcard)：用来匹配值的一部分的特殊字符

2、搜搜模式(search pattern)：由字面值、通配符或两者组合构成的搜索条件

3、**为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

4、谓词：操作符何时不是操作符?答案是在它作为谓词(predi- cate)时。从技术上说，LIKE 是谓词而不是操作符。虽然最终的结果是相同的，但应该对此术语有所了解，以免在 SQL 文档中遇到此术语时不知道

8.1.1. 百分号(%)通配符

1、%表示任何字符出现任意次数(包括 0 次，有点像正则表达式中的".*"以及 bash 中的通配符"*")

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '%anvil%';
```

2、注意 NULL：虽然似乎%通配符可以匹配任何东西，但有一个例外，即 NULL。即使是 WHERE prod_name LIKE '%'也不能匹配 用值 NULL 作为产品名的行

8.1.2. 下划线(_)通配符

1、下划线只匹配单个字符而不是多个字符(有点像正则表达式中的"."和 bash 中的通配符"?")

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '_ ton anvil';
```

8.2. 使用通配符的技巧

1、MySQL 的通配符很有用。但这种功能是有代价的：通配符搜索的处理一般要比前面讨论的其他搜索所花时间更长

- 不要过度使用通配符。如果其他操作符能达到相同的目的，应该使用其他操作符
- 在确实需要使用通配符时，除非绝对有必要，否则不要把它们用在搜索模式的开始处。把通配符置于搜索模式的开始处，搜索起来是最慢的
- 仔细注意通配符的位置。如果放错地方，可能不会返回想要的数

Chapter 9. 用正则表达式进行搜索

9.1. 正则表达式介绍

1、正则表达式用正则表达式语言来建立，正则表达式语言是用来完成刚讨论的所有工作以及更多工作的一种特殊语言。与任意语言一样，正则表达式具有你必须学习的特殊的语法和指令

9.2. 使用 MySQL 正则表达式

1、MySQL 用 WHERE 子句对正则表达式提供了初步的支持，允许你指定正则表达式，过滤 SELECT 检索出的数据

2、MySQL 仅支持多数正则表达式实现的一个很小的子集

9.2.1. 基本字符匹配

1、为在搜索子句中使用通配符，必须使用 REGEXP 操作符，它告诉 MySQL:REGEXP 后所跟的东西作为正则表达式

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '1000'
ORDER BY prod_name;
```

4、LIKE 与 REGEXP 的区别

- LIKE 匹配整个列。如果被匹配的文本在列值中(而并非匹配整个列)出现，LIKE 将不会找到它，相应的行也不被返回
- 而 REGEXP 在列值内进行匹配，如果被匹配的文本在列值中出现，REGEXP 将会找到它，相应的行将被返回

5、MySQL 中的正则表达式匹配(自版本 3.23.4 后)不区分大小写(即，大写和小写都匹配)

- 为区分大小写，可使用 BINARY 关键字

```
WHERE prod_name REGEXP BINARY 'JetPack .000'
```

9.2.2. 进行 OR 匹配

1、"|"

```
SELECT prod_name
FORM products
WHERE prod_name REGEXP '1000|2000'
ORDER BY prod_name;
```

2、支持两个以上的"|"一起使用

9.2.3. 匹配几个字符之一

1、[]：匹配任何单一字符

- []是另一种形式的 OR 语句

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '[123] Ton'
ORDER BY prod_name;
```

2、否定形式：[^]

9.2.4. 匹配范围

1、集合可用来定义要匹配的一个或多个字符，可用"-"来定义一个范围

- [123456789]等价于[1-9]

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '[1-5] Ton'
ORDER BY prod_name;
```

9.2.5. 匹配特殊字符

1、为了匹配特殊字符，必须用\\为前导，正则表达式内具有特殊意义的所 有字符都必须以这种方式转义

```
SELECT vend_name
FROM vendors
WHERE vend_name REGEXP '\\.'
ORDER BY vend_name;
```

2、\\也用来引用元字符(具有特殊含义的字符)

- \\f 换页
- \\n 换行
- \\r 回车
- \\t 制表
- \\v 纵向制表

3、为了匹配反斜杠"\"本身，需要使用"\\\"

4、多数正则表达式实现使用单个反斜杠转义特殊字符以便能使用这些字符本身，但 MySQL 要求两个反斜杠(MySQL 自己解释一个，正则表达式库解释另一个)

9.2.6. 匹配字符类

1、存在找出你自己经常使用的数字、所有字母字符或所有数字字母字符等的匹配

2、常用字符类

- [:alnum:]：任意字母和数字，同[a-zA-Z0-9]
- [:alpha:]：任意字符(同[a-zA-Z])
- [:blank:]：空格和制表(同[\\t])
- [:cntrl:]：ASCII 控制字符(ASCII0 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:]：任意十六进制数字，同[a-fA-F0-9]
- [:xdigit:]

9.2.7. 匹配多个实例

1、正则表达式重复元字符

- *：0 个或多个匹配
- +：1 个或多个匹配
- ?：0 个或 1 个匹配
- {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

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '\\([0-9] sticks?\\)'
ORDER BY prod_name;

SELECT prod_name
FROM products
WHERE prod_name REGEXP '[[:digit:]]{4}'
ORDER BY prod_name;
```

9.2.8. 定位符

1、定位符

- ^：文本的开始(注意与[^]中的^含义不同)
- $：文本的结尾
- [[:<:]]：词的开始
- [[:>:]]：词的结尾

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '^[0-9\\.]'
ORDER BY prod_name;
```

9.2.9. 简单的正则表达式测试

1、可以在不使用数据库表的情况下用 SELECT 来测试正则表达式。REGEXP 检查总是返回 0(没有匹配)或 1(匹配)。可以用带文字串的 REGEXP 来测试表达式，并试验它们

```
SELECT 'hello' REGEXP '[0-9]';
```

Chapter 10. 创建计算字段

10.1. 计算字段

1、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，我们需要直接从数据库中检索出转换、计算或格式化过的数据，而不是检索出数据，然后再在客户机应用程序或报告程序中重新格式化

2、计算字段并不实际存在于数据库表中，而是运行时在 SELECT 语句内创建的

3、可在 SQL 语句内完成的许多转换和格式化工作都可以直接在客户机应用程序内完成，但一般来说，在数据库服务器上完成这些操作比在客户机中完成要亏爱的多，因为 DBMS 是设计来快速有效的完成这种处理的

10.2. 拼接字段

1、在 MySQL 的 SELECT 语句中，可使用 Concat()函数来拼接两个列

- 多数 DBMS 使用"+"或"||"来实现拼接，MySQL 则使用 Concat()函数来实现

```
SELECT Concat(vend_name,'(',vend_country,')')
FROM vendors
ORDER BY vend_name;
```

- Concat()拼接串，即把多个串连接起来形成一个较长的串，各个串之间用逗号分隔

2、使用别名

- SELECT 配合 Concat()拼接字段很简单，但是该新计算列的名字是未定义的
- 一个未命名的列不能用于客户机应用中
- 为了解决这个问题，SQL 支持列别名，别名是一个字段或值的替换名，别名用 AS 关键字赋予

```
SELECT Concat(RTrim(vend_name),'(',RTrim(vend_country),')') AS
vend_title
FROM vendors
ORDER BY vend_name;
```

- 其中 RTrim()去掉串右边的空格
- 同理 LTrim()去掉串左边的空格
- Trim()去掉串左右两边的空格

10.3. 执行算数计算

1、计算字段的另一个常见用途是对检索出的数据进行算数计算

```
SELECT prod_id,
       quantity,
       item_price
       quantity*item_price AS expanded_price
FROM orderitems
WHERE order_num=20005;
```

Chapter 11. 使用数据处理函数

11.1. 函数

1、与其他大多数计算机语言一样，SQL 支持利用函数来处理数据，函数一般是在数据上执行的，它给数据的转换和处理提供了方便

2、能运行在多个系统上的代码称为可移植的(portable)

- 相对来说 SQL 语句是可移植的，在 SQL 实现之间有差异时，这些差异通常比较容易处理
- 函数的可移植性却不强，几乎每种主要的 DBMS 的实现都支持其他实现不支持的函数，而且有时差别很大
- 如果决定使用函数，应该保证做好代码注释，以便以后你(或其他人)能确切知道所编写 SQL 代码的含义

11.2. 使用函数

1、大多数 SQL 实现支持以下类型的函数

- 用于处理文本串(如删除或填充值，转换值为大写或小写)的文本函数
- 用于在数值数据上进行算数操作(如返回绝对值，进行代数运算)的数值函数
- 用于处理日期和时间值并从这些值中提取特定成分(例如，返回两个日期的差，检查日期的有效性等)的日期函数
- 返回 DBMS 正使用的特殊信息(如返回用户登陆信息，检查版本细节)的系统函数

11.2.1. 文本处理函数

1、常用的文本处理函数

- 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

11.2.2. 日期和时间处理函数

1、日期和时间采用相应的数据类型和特殊的格式存储，以便能快速和有效地排序或过滤，并且节省物理存储空间

2、一般，应用程序不使用用来存储日期和时间的格式，因此日期和时间函数总是被用来读取、统计和处理这些值，于是日期和时间函数在 MySQL 中具有重要作用

3、常用的日期和时间处理函数

- 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

4、应该总是使用 4 位数字的年份，避免 MySQL 做出任何假定

5、MySQL 使用的日期格式：yyyy--mm-dd，避免二义性!!!

6、例子 1

```
SELECT cust_id,order_num
FROM orders
WHERE Date(order_date)='2005-09-01';

SELECT cust_id,order_num
FROM orders
WHERE order_date='2005-09-01';
```

- 下面这种无法匹配 2005-09-01 11:30:05 这样的日期时间

7、例子 2

- 想要检索出 2005 年 9 月的所有订单

```
SELECT cust_id,order_num
FROM orders
WHERE Date(order_date) BETWEEN '2005-09-01' AND '2005-09-30';
```

- 另一种方法，不需要记得每个月到底有几天

```
SELECT cust_id,order_num
FROM orders
WHERE Year(order_date)=2005 AND Month(order_date)=9;
```

11. 2. 3. 数值处理函数

1、数值处理函数仅处理数值数据，这些函数一般主要用于代数运算、三角或集合运算，因此没有串货日期-时间处理函数使用得那么频繁

2、常用的数值处理函数

- 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

Chapter 12. 汇总数

12.1. 聚集函数

1、我们经常需要汇总数据而不用把它们实际检索出来，为此 MySQL 提供了专门的函数，使用这些函数，MySQL 查询可用于检索数据，以便分析和报表生成，这种类型的检索例子有如下几种

- 确定表中行数
- 获得表中行组的和
- 找出表列(或所有行货某些特定行)的最大值、最小值和平均值

2、聚集函数：运行在行组上，计算和返回单个值的函数

- 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

12.1.1. AVG()函数

1、AVG()通过对表中行数计算并计算特定列值之和，求得该列的平均值

- AVG()可用来返回所有列的平均值，也可以用来返回特定列或行的平均值

```
SELECT AVG(prod_price) AS avg_price
FROM products;

SELECT AVG(prod_price) AS avg_price
FROM products
WHERE vend_id=1003;
```

2、AVG()只能用来确定特定数值列的平均值，而且列
必须作为函数参数给出，为了获取多个列的平均值，必须使用多个 AVG()函数
3、NULL 值：AVG()函数忽略列值为 NULL 的行

12.1.2. COUNT()函数

1、COUNT()函数进行计算，可利用 COUNT()确定表中行的数目或符合特定条件的行的数目
2、COUNT()函数有两种使用方式

- 使用 COUNT(*)对表中行的数目进行计数，不管表列中包含的是空值(NULL)还是非空值
- 使用 COUNT(column)对特定列中具有值的行进行计数，忽略 NULL 值

```
SELECT COUNT(cust_email) AS num_cust
FROM customers;
```

12.1.3. MAX()函数

1、MAX()返回指定列中的最大值，MAX()要求指定列名

- MAX()一般用来找出最大的数值或日期，但 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
- MAX()函数忽略值为 NULL 的行

12.1.4. MIN()函数

1、与 MAX()类似，含义相反

12.1.5. SUM()函数

1、SUM()函数用来返回指定列的和(总计)

```
SELECT SUM(quantity) AS items_ordered
FROM orderitems
WHERE order_num=20005;

SELECT SUM(item_price*quantity) AS total_price
FROM orderitems
WHERE order_num=20005;
```

2、利用标准的算数操作符，所有聚集函数都可用来执行多个列上的计算

3、SUM()函数忽略列值为 NULL 的行

12.2. 聚集不同值(UNIQUE)

1、上述五个聚集(AVG/COUNT/MAX/MIN/SUM)函数都可以如下使用：

- 对所有的执行计算，指定 ALL 参数或不给参数(因为 ALL 是默认的行为)
- 只包含不同的值，指定 DISTINCT 参数

```
SELECT AVG(DISTINCT prod_price) AS avg_price
FROM products
WHERE vend_id=1003;
```

2、注意，DISTINCT 只能用于 COUNT()，而不能用于 COUNT(*)

12.3. 组合聚集函数

1、目前为止的所有聚集函数例子都只涉及单个函数，但实际上 SELECT 语句可根据需要包含多个聚集函数

```
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;
```

Chapter 13. 分组数据

13.1. 数据分组

1、分组允许把数据分成多个逻辑组，以便能对每个组进行聚集计算

13.2. 创建分组

1、分组是 SELECT 语句的 GROUP BY 子句中建立的

```
SELECT vend_id,COUNT(*) AS num_prods
FROM products
GROUP BY vend_id
```

- GROUP BY 子句指示 MySQL 按 vend_id 排序并分组数据，这导致对每个 vend_id 而不是整个表计算 num_prods 一次

2、使用 GROUP BY，就不必指定要计算和估值的每个组，系统会自动完成

3、重要的规定

- 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， 为数据分组提供更细致的控制
- 如果在 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。换句话说，在建立分组时，指定的所有列都一起计算(所以不能从个别的列取回数据)
- 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)。**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**
- 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
- 如果分组列中具有 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
- GROUP BY 子句必须出现在 WHERE 子句之后，ORDER BY 子句之前

13.3. 过滤分组

1、除了能用 GROUP BY 分组数据外，MySQL 还允许过滤分组，规定包括哪些分组，排除哪些分组

2、WHERE 和 HAVING

- HAVING 非常类似于 WHERE
- 唯一的差别是 WHERE 过滤行，而 HAVING 过滤分组
- 这里有另一种理解方法，WHERE 在数据分组前进行过滤，HAVING 在数据分组后进行过滤。这是一个重要的区别，WHERE 排除的行不包括在分组中。这可能会改变计算值，从而影响 HAVING 子句中基于这些值过滤掉的分组

```
SELECT cust_id, COUNT(*) AS orders
FROM orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;

SELECT cust_id, COUNT(*) AS orders
```

```
FROM orders
WHERE prod_price >= 10
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```

13.4. 分组和排序

1、ORDER BY 与 GROUP BY 的差别

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

2、一般在使用 GROUP BY 子句时，应该也给出 ORDER BY 子句。这是保证数据正确排序的唯一方法。千万不要仅依赖 GROUP BY 排序数据

```
SELECT order_num ,SUM(quantity*item_price) AS ordertotal
FROM orderitems
GROUP BY order_num
HAVING SUM(quantity*item_price) >= 50;
```

13.5. SELECT 子句顺序

1、SELECT 子句及其顺序

- SELECT：要返回的列或表达式，必须使用
- 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，可选
- GROUP BY：分组说明，仅在按组计算聚集时使用
- HAVING：组级过滤，可选
- ORDER BY：输出排序顺序，可选
- LIMIT：要检索的行数，可选

Chapter 14. 使用子查询

14.1. 子查询

1、SELECT 语句是 SQL 的查询

2、查询：任何 SQL 语句都是查询，但此属于一般指 SELECT 语句

3、SQL 还允许创建子查询(subquery)，即嵌套在其他查询中的查询

14.2. 利用子查询进行过滤

1、现在需要列出订购物品 TNT2 的所有客户

1) 检索包含物品 TNT2 的所有订单编号
2) 检索具有前一步骤列出的订单编号的所有客户 ID
3) 检索前一步骤返回的所有客户 ID 的客户信息

- 上述每个步骤都可以单独作为一个查询来执行，可以把一条 SELECT 语句返回的结果用于另一条 SELECT 语句的 WHERE 子句
- 也可以使用子查询来把三个查询组合成一条语句

```
SELECT order_num
FROM orderitems
WHERE prod_id='TNT2'

SELECT cust_id
FROM orders
WHERE order_num IN (20005,20007)   <==20005 20007 为前一步结果

SELECT cust_id
FROM orders
WHERE order_num IN (SELECT order_num
                    FROM orderitems
                    WHERE prod_id='TNT2');
```

2、在 SELECT 语句中，子查询总是从内向外处理

```
SELECT curst_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='TNT2'));
```

14.3. 作为计算字段使用子查询

1、使用子查询的另一方法是创建计算字段

2、假设需要显示 customers 表中每个客户的订单总数

- 从 customers 表中检索客户列表
- 对于检索出的每个客户，统计其在 orders 表中的订单数目

```
SELECT cust_name,
       cust_state,
```

```
    (SELECT COUNT(*)
    FROM orders
    WHERE orders.cust_id=customers.cust_id) AS orders
FROM customers
ORDER BY cust_name;
```

- 从 orders 表中查询每个用户的订单总数

Chapter 15. 联结表

15.1. 联结

1、SQL 最强大的功能之一就是能在数据检索查询的执行中联结(join)表。联结是利用 SQL 的 SELECT 能执行的最重要的操作，很好地理解联结及其语法是学习 SQL 的一个极为重要的组成部分

15.1.1. 关系表

1、现在，假如有由同一供应商生产的多种物品，那么在何处存储供应商信息(如，供应商名、地址、联系方法等)呢?将这些数据与产品信息分开存储的理由如下

- 因为同一供应商生产的每个产品的供应商信息都是相同的，对每个产品重复此信息既浪费时间又浪费存储空间
- 如果供应商信息改变(例如，供应商搬家或电话号码变动)，只需改动一次即可
- 如果有重复数据(即每种产品都存储供应商信息)，很难保证每次输入该数据的方式都相同。不一致的数据在报表中很难利用

2、关键是，相同数据出现多次决不是一件好事，此因素是关系数据库设计的基础。关系表的设计就是要保证把信息分解成多个表，一类数据一个表。各表通过某些常用的值(即关系设计中的关系(relational))互相关联

- 在这个例子中，可建立两个表，一个存储供应商信息，另一个存储产品信息。vendors 表包含所有供应商信息，每个供应商占一行，每个供应商具有唯一的标识
- products 表只存储产品信息，它除了存储供应商 ID(vendors 表的主键)外不存储其他供应商信息。vendors 表的主键又叫作 products 的外键，它将 vendors 表与 products 表关联，利用供应商 ID 能从 vendors 表中找出相应供应商的详细信息

3、**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**

4、这样做的好处

- 供应商信息不重复，从而不浪费时间和空间
- 如果供应商信息变动，可以只更新 vendors 表中的单个记录，相关表中的数据不用改动
- 由于数据无重复，显然数据是一致的，这使得处理数据更简单
- 总之，关系数据可以有效地存储和方便地处理。因此，关系数据库 的可伸缩性远比非关系数据库要好

15.1.2. 为什么要使用联结

1、联结：联结是一种机制，用来在一条 SELECT 语句中关联表，因此称之为联结

- 联结不是物理实体。换句话说，它在实际的数据库表中不存在。联结由 MySQL 根据需要建立，它存在于查询的执行当中

15.2. 创建联结

1、例子

```
SELECT vend_name,prod_name,prod_price
FROM vendors,products
WHERE vendors.vend_id=products.vend_id
ORDER BY vend_name,prod_name;
```

2、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)。如果引用一个 没有用表名限制的具有二义性的列名，MySQL 将返回错误

15.2.1. WHERE 子句的重要性

1、在一条 SELECT 语句中联结几个表时，相应的关系是在运行中构造的。在数据库表的定义中不存在能指示 MySQL 如何对表进行联结的东西

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、没有 WHERE 子句，**第一个表中的每个行将与第二个表中的每个行配对**，而不管它们逻辑上是否可以配在一起

- 取出第一个表的第一行，然后与第二个表所有行匹配一次
- 取出第一个表的第二行，然后与第二个表的所有行匹配一次
- ...
- 取出第一个表的第 n 行，然后与第二个表的所有行匹配一次
- **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**

15.2.2. 内部联结

1、目前为止所用的联结称为等值联结(equijoin)，它基于两个表之间的相等测试。这种联结也称为内部联结，对于这种联结可以使用稍 微不同的语法来明确指定联结的类型

```
SELECT vend_name,prod_name,prod_price
FROM vendors INNER JOIN products
ON vendors.vend_id=products.vend_id;
```

2、ANSI SQL 规范首选 INNER JOIN 语法。此外，尽管使用 WHERE 子句定义联结的确比较简单，但是使用明确的联结语法能够确保不会忘记联结条件，有时候这样做也能影响性能

15.2.3. 联结多个表

1、SQL 对一条 SELECT 语句中可以联结的表的数目没有限制，创建联结的基本规则也相同。首先列出所有表，然后定义表之间的关系

```
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=20005;
```

Chapter 16. 创建高级联结

16.1. 使用表别名

1、别名除了用于列名和计算字段外，SQL 还允许给表名起别名。这样做有两个主要理由：

- 缩短 SQL 语句
- 允许在单条 SELECT 语句中多次使用相同的表

```
SELECT cust_name,cust_contact
FROM customers AS c, orders AS o, ordertimes AS oi
WHERE c.cust_id=o.cust_id
   AND oi.order_num=o.order_num
   AND prod_id='TNT2';
```

16.2. 使用不同类型的联结

1、迄今为止，我们使用的只是称为内部联结或等值联结(equijoin)的简单联结。现在来看 3 种其他联结，它们分别是自联结、自然联结和外部联结

16.2.1. 自联结

1、以一个例子说明

- 假如你发现某物品(其 ID 为 DTNTR)存在问题，因此想知道生产该物品的供应商生产的其他物品是否也存在这些问题。此查询要求首先找到生产 ID 为 DTNTR 的物品的供应商，然后找出这个供应商生产的其他物品
- 以下是用子查询的解决方案

```
SELECT prod_id, prod_name
FROM products
WHERE vend_id=(SELECT vend_id
                      FROM products
                      WHERE prod_id='DTNTR');
```

- 以下是用联结的方案

```
SELECT p1.prod_id, p1.prod_name
FROM products AS p1, products AS p2
WHERE p1.vend_id=p2.vend_id
   AND p2.prod_id='DTNTR';
```

  - 此次查询的两个表实际上是同一张表，但是引用列名仍然具有二义性因为不知道哪张表
  - 为了解决这个二义性问题，使用了别名

2、用自联结而不用子查询自联结通常作为外部语句用来替代从相同表中检索数据时使用的子查询语句。虽然最终的结果是相同的，但有时候处理联结远比处理子查询快得多。应该试一下两种方法，以确定哪一种的性能更好

16.2.2. 自然联结

1、无论何时对表进行联结，应该至少有一个列出现在不止一个表中(被联结的列)。标准的联结(前一章中介绍的内部联结)返回所有数据，甚至相同的列多次出现。自然联结排除多次出现，使每个列只返回一次

2、自然联结是这样一种联结，其中你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
3、<未完成>：不懂

16.2.3. 外部联结

1、许多联结将一个表中的行与另一个表中的行相关联

```
SELECT customers.cust_id, orders.order_num
FROM customers INNER JOIN orders
   ON customers.cust_id=orders.cust_id;

SELECT customers.cust_id,orders.order_num
FROM customers LEFT OUTER JOIN orders
ON customers.cust_id=orders.cust_id;
```

- 与内部联结不同的是，外部联结还包括没有关联行的行
- 在使用 OUTER JOIN 语法时，必须使用 RIGHT 或 LEFT 关键字指定包括其所有行的表(RIGHT 指出的是 OUTER JOIN 右边的表，而 LEFT 指出的是 OUTER JOIN 左边的表)

2、<未完成>：不懂

16.3. 使用带聚集函数的联结

1、聚集函数可以与联结一起使用

```
SELECT customers.cust_name,
       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;
```

16.4. 使用联结和联结条件

1、联结及其使用的要点

- 注意所使用的联结类型。一般我们使用内部联结，但使用外部联结也是有效的
- 保证使用正确的联结条件，否则将返回不正确的数据
- 应该总是提供联结条件，否则会得出笛卡儿积
- 在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。虽然这样做是合法的，一般也很有用，但应该在一起测试它们前，分别测试每个联结。这将使故障排除更为简单

Chapter 17.组合查询

17.1. 组合查询

1、多数 SQL 查询都只包含从一个或多个表中返回数据的单条 SELECT 语句。MySQL 也允许执行多个查询(多条 SELECT 语句)，并将结果作为单个查询结果集返回。这些组合查询通常称为并(union)或复合查询(compound query)

2、有两种基本情况，其中需要使用组合查询

- 在单个查询中从不同的表返回类似结构的数据
- 对单个表执行多个查询，按单个查询返回数据

3、组合查询和多个 WHERE 条件：多数情况下，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。换句话说，任何具有多个 WHERE 子句的 SELECT 语句都可以作为一个组合查询给出

17.2. 创建组合查询

1、可用 UNION 操作符来组合数条 SQL 查询。利用 UNION，可给出多条 SELECT 语句，将它们的结果组合成单个结果集

17.2.1. 使用 UNION

1、UNION 的使用很简单。所需做的只是给出每条 SELECT 语句，在各条语句之间放上关键字 UNION

```
SELECT vend_id,prod_id,prod_price
FROM products
WHERE prod_price<=5;

SELECT vend_id,prod_id,prod_price
FROM products
WHERE vend_id IN(1001,1002);
```

- 对于以上两个查询，可以使用 UNION 语句

```
SELECT vend_id,prod_id,prod_price
FROM products
WHERE prod_price<=5
UNION
SELECT vend_id,prod_id,prod_price
FROM products
WHERE vend_id IN(1001,1002)
```

- 也可以使用 WHERE 子句

```
SELECT vend_id,prod_id,prod_price
FROM products
WHERE prod_price<=5
   OR vend_id IN(1001,1002);
```

17.2.2. UNION 规则

1、注意以下几条规则

- 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(因此，如果组合 4 条 SELECT 语句，将要使用 3 个 UNION 关

键字)

- 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)
- 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(例如，不同的数值类型或不同的日期类型)

17. 2. 3. 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(换句话说，它的行为与单条 SELECT 语句中使用多个 WHERE 子句条件一样)

2、这是 UNION 的默认行为，但是如果需要，可以改变它。事实上，如果想返回所有匹配行，可使用 UNION ALL 而不是 UNION

3、UNION 与 WHERE：

- UNION 几乎总是完成与多个 WHERE 条件相同的工作
- UNION ALL 为 UNION 的一种形式，它完成 WHERE 子句完成不了的工作
- 如果确实需要每个条件的匹配行全部出现(包括重复行)，则必须使用 UNION ALL 而不是 WHERE

17. 2. 4. 对组合查询结果排序

1、SELECT 语句的输出用 ORDER BY 子句排序。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。**对于结果集，不存在用一种方式排序一部分，而又用另一种方式排序另一部分的情况**，因此不允许使用多条 ORDER BY 子句

Chapter 18. 全文本搜索

18.1. 理解全文本搜索

1、并非所有引擎都支持全文本搜索

- MySQL 支持几种基本的数据库引擎。并非所有的引擎都支持本书所描述的全文本搜索
- 两个最常使用的引擎为 MyISAM 和 InnoDB，前者支持全文本搜索，而后者不支持。这就是为什么虽然本书中创建的多数样例表使用 InnoDB，而有一个样例表(productnotes 表)却使用 MyISAM 的原因。如果你的应用中需 要全文本搜索功能，应该记住这一点

2、LIKE 关键字配合通配符以及 REGEX 关键字配合正则表达式，可以完成非常复杂的匹配模式，但是存在几个重要的限制

- 性能：通配符和正则表达式匹配通常要求 MySQL 尝试匹配表中所有行(而且这些搜索极少使用表索引)。因此，由于被搜索行数不断增加，这些搜索可能非常耗时
- 明确控制：使用通配符和正则表达式匹配，很难(而且并不总是能)明确地控制匹配什么和不匹配什么。例如，指定一个词必须匹配，一个词必须不匹配，而一个词仅在第一个词确实匹配的情况下才可以匹配或者才可以不匹配
- 智能化的结果：虽然基于通配符和正则表达式的搜索提供了非常灵活的搜索，但它们都不能提供一种智能化的选择结果的方法。例如，一个特殊词的搜索将会返回包含该词的所有行，而不区分包含单个匹配的行和包含多个匹配的行(按照可能是更好的匹配来排列它们)。类似，一个特殊词的搜索将不会找出不包含该词但包含其他相关词的行

3、所有这些限制以及更多的限制都可以用全文本搜索来解决。在使用全文本搜索时，MySQL 不需要分别查看每个行，不需要分别分析和处理每个词。MySQL 创建指定列中各词的一个索引，搜索可以针对这些词进行。这样，MySQL 可以快速有效地决定哪些词匹配(哪些行包含它们)，哪些词不匹配，它们匹配的频率，等等

18.2. 使用全文本搜索

1、为了进行全文本搜索，必须索引被搜索的列，而且要随着数据的改变不断地重新索引。在对表列进行适当设计后，MySQL 会自动进行所有的索引和重新索引

18.2.1. 启用全文本搜索支持

1、一般在创建表时启用全文本搜索。CREATE TABLE 语句接受 FULLTEXT 子句，它给出被索引列的一个逗号分隔的列表

```
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;
```

2、不要在导入数据时使用 FULLTEXT：更新索引要花时间，虽然不是很多，但毕竟要花时间。如果正在导入数据到一个新表，此时不应该启用 FULLTEXT 索引。应该首先导入所有数据，然后再修改表，定义 FULLTEXT。这样有助于更快地导入数据

18. 2. 2. 进行全文本搜索

1、在索引之后，使用两个函数 Match()和 Against()执行全文本搜索，其中 Match()指定被搜索的列，Against()指定要使用的搜索表达式

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('rabbit');
```

2、使用完整的 Match()说明

- 传递给 Match()的值必须与 FULLTEXT()定义中的相同
- 如果指定多个列，则必须列出它们(而且次序正确)

3、搜索不区分大小写

- 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

4、Match()和 Against()用来建立一个计算列(别名为 rank)，此列包含全文本搜索计算出的等级值。等级由 MySQL 根据行中词的数目、唯一词的数目、整个索引中词的总数以及包含该词的行的数目计算出来

- 如果指定多个搜索项，则包含多数匹配词的那些行将具有比包含较少词(或仅有一个匹配)的那些行高的等级值

18. 2. 3. 使用查询扩展

1、查询扩展用来设法放宽所返回的全文本搜索结果的范围

- 例如，你想找出所有提到 anvils 的注释。只有一个注释包含词 anvils，但你还想找出可能与你的搜索有关的所有其他行，即使它们不包含词 anvils

2、在使用查询扩展时，MySQL 对数据和索引进行两遍扫描来完成搜索

- 首先，进行一个基本的全文本搜索，找出与搜索条件匹配的所有行
- 其次，MySQL 检查这些匹配行并选择所有有用的词
- 再其次，MySQL 再次进行全文本搜索，这次不仅使用原来的条件，而且还使用所有有用的词

3、<未完成>

18. 2. 4. 布尔文本搜索

1、MySQL 支持全文本搜索的另外一种形式，称为布尔方式(boolean mode)。以布尔方式，可以提供关于如下内容的细节

- 要匹配的词
- 要排斥的词(如果某行包含这个词，则不返回该行，即使它包其他指定的词也是如此)
- 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)
- 表达式分组

- 另外一些内容

2、即使没有 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。但这是一种非常缓慢的操作

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('heavy' IN BOOLEAN MODE)
```

3、全文本布尔操作符

- +：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)
- ~：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)

4、例子说明

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('+rabbit +bait' IN BOOLEAN MODE);
```

- 这个搜索匹配包含词 rabbit 和 bait 的行

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('rabbit bait' IN BOOLEAN MODE);
```

- 没有指定操作符，这个搜索匹配包含 rabbit 和 bait 中的至少一 个词的行

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('"rabbit bait"' IN BOOLEAN MODE);
```

- 这个搜索匹配短语 rabbit bait 而不是匹配两个词 rabbit 和 bait

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('>rabbit <carrot' IN BOOLEAN MODE);
```

- 匹配 rabbit 和 carrot，增加前者的等级，降低后者的等级

```
SELECT note_text
FROM productnotes
WHERE Match(note_text) Against('+safe +(<combination)' IN BOOLEAN MODE);
```

- 这个搜索匹配词 safe 和 combination，降低后者的等级

5、排列而不排序：在布尔方式中，不按等级值降序排序返回的行

18.2.5. 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。如果需要，可以覆盖这个列表(请参阅 MySQL 文档以了解如何完成此工作)
3、许多词出现的频率很高，搜索它们没有用处(返回太多的结果)。因此，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。50%规则不用于 IN BOOLEAN MODE
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。例如，don't 索引为 dont
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
7、如前所述，仅在 MyISAM 数据库引擎中支持全文本搜索

Chapter 19. 插入数据

19.1. 数据插入

1、顾名思义，INSERT 是用来插入(或添加)行到数据库表的。插入可以用几种方式使用

- 插入完整的行
- 插入行的一部分
- 插入多行
- 插入某些查询的结果

19.2. 插入完整的行

1、把数据插入表中的最简单的方法是使用基本的 INSERT 语法，它要求指定表名和被插入到新行中的值

```
INSERT INTO Customers
VALUES(NULL,
       'Pep E. LaPew',
       '100 Main Street',
       'Los Angeles',
       'CA',
       '90046',
       'USA',
       NULL,
       NULL);
```

- 存储到每个表列中的数据在 VALUES 子句中给出，对每个列必须提供一个值
- 如果某个列没有值(如上面的 cust_contact 和 cust_email 列)，应该使用 NULL 值(假定表允许对该列指定空值)
- 各个列必须以它们在表定义中出现的次序填充
- 虽然这种语法很简单，但并不安全，应该尽量避免使用
- 上述语句语句高度依赖于表中列的定义次序，并且还依赖于其次序容易获得的信息

2、更安全的方法如下

```
INSERT INTO Customers(cust_name,
       cust_address,
       cust_city,
       cust_state,
       cust_zip,
       cust_country,
       cust_contact,
       cust_email)
VALUES('Pep E. LaPew',
       '100 Main Street',
       'Los Angeles',
       'CA',
       '90046',
       'USA',
```

NULL,
NULL);

3、**总是使用列的列表**：一般不要使用没有明确给出列的列表的 INSERT 语句。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

4、**仔细地给出值**：不管使用哪种 INSERT 语法，都必须给出 VALUES 的正确数目。如果不提供列名，则必须给每个表列提供一个值。如果提供列名，则必须对每个列出的列给出一个值。如果不这样，将产生一条错误消息，相应的行插入不成功

5、**省略列**：如果表的定义允许，则可以在 INSERT 操作中省略某些列。省略的列必须满足以下某个条件。

- 该列定义为允许 NULL 值(无值或空值)
- 在表定义中给出默认值。这表示如果不给出值，将使用默认值
- 如果对表中不允许 NULL 值且没有默认值的列不给出值，则 MySQL 将产生一条错误消息，并且相应的行插入不成功

19. 2. 1. 插入多个行

1、可以使用多条 INSERT 语句，甚至一次提交它们，每条语句用一个分号结束

```
INSERT INTO customers(
    ...
    )
VALUES(
    ...
    );
INSERT INTO customers(
    ...
    )
VALUES(
    ...
    );
```

2、或者，只要每条 INSERT 语句中的列名(和次序)相同

```
INSERT INTO customers(
    ...
    )
VALUES(
    ...
    ),
VALUES(
    ...
    );
```

19. 3. 插入检索出的数据

1、INSERT 一般用来给表插入一个指定列值的行。但是，INSERT 还存在另一种形式，可以利用它将一条 SELECT 语句的结果插入表中。这就是所谓的 INSERT SELECT，顾名思义，它是由一条 INSERT 语句和一条 SELECT 语句组成的

```
INSERT INTO customers(cust_id,
        cust_constact,
        cust_email,
        cust_name,
        cust_address,
        cust_city,
        cust_state,
        cust_zip,
        cust_country)
SELECT cust_id,
        cust_constact,
        cust_email,
        cust_name,
        cust_address,
        cust_city,
        cust_state,
        cust_zip,
        cust_country
FROM custnew;
```

2、INSERT SELECT 中的列名：为简单起见，这个例子在 INSERT 和 SELECT 语句中使用了相同的列名。但是，不一定要求列名匹配

- 事实上，MySQL 甚至不关心 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，第二列将用来填充表列中指定的第二个列，如此等等
- 这对于从使用不同列名的表中导入数据是非常有用的

Chapter 20. 更新和删除数据

20.1. 更新数据

1、为了更新(修改)表中的数据，可使用 UPDATE 语句。可采用两种方式使用 UPDATE

- 更新表中特定行
- 更新表中所有行

2、不要省略 WHERE 子句：在使用 UPDATE 时一定要注意细心。因为稍不注意，就会更新表中所有行

3、UPDATE 语句非常容易使用，甚至可以说是太容易使用了。基本的 UPDATE 语句由 3 部分组成，分别是

- 要更新的表
- 列名和它们的新值
- 确定要更新行的过滤条件

```
UPDATE customers
SET cust_email='elmer@fudd.com'
WHERE cust_id=10005;
```

4、在更新多个列时，只需要使用单个 SET 命令，每个"列=值"对之间用逗号分隔(最后一列之后不用逗号)

```
UPDATE customers
SET cust_name='The Fudds',
      cust_email='elmer@fudd.com'
WHERE cust_id=10005;
```

5、为了删除某个列的值，可设置它为 NULL(假如表定义允许 NULL 值)

```
UPDATE customers
SET cust_email=NULL
WHERE cust_id=10005;
```

20.2. 删除数据

1、为了从一个表中删除(去掉)数据，使用 DELETE 语句。可以两种方式使用 DELETE

- 从表中删除特定行
- 从表中删除所有行

2、不要省略 WHERE 子句：在使用 DELETE 时一定要注意细心。因为稍不注意，就会错误地删除表中所有行

```
DELETE FROM customer
WHERE cust_id=10006;
```

3、DELETE 不需要列名或通配符，DELETE 删除整行而不是删除列，若要删除列(赋值为 NULL)请使用 UPDATE

4、删除表的内容而不是表：DELETE 语句从表中删除行，甚至是删除表中所有行。但是，DELETE 不删除表本身

5、更快的删除：如果想从表中删除所有行，不要使用 DELETE。可使用 TRUNCATE TABLE 语句，它完成相同的工作，但速度更快(TRUNCATE 实际是删除原来的表并重新创建一个表，而不是逐行删除表中的数据)

20.3. 更新和删除的指导原则

1、UPDATE 和 DELETE 语句全都具有 WHERE 子句，这样做的理由很充分。如果省略了 WHERE 子句，则 UPDATE 或 DELETE 将被应用到表中所有的行。换句话说，如果执行 UPDATE 而不带 WHERE 子句，则表中每个行都将用新值更新。类似地，如果执行 DELETE 语句而不带 WHERE 子句，表的所有数据都将被删除

2、下面是许多 SQL 程序员使用 UPDATE 或 DELETE 时所遵循的习惯

- 除非确实打算更新和删除每一行，否则绝对不要使用不带 WHERE 子句的 UPDATE 或 DELETE 语句
- 保证每个表都有主键(如果忘记这个内容，请参阅第 15 章)，尽可能 像 WHERE 子句那样使用它(可以指定各主键、多个值或值的范围)
- 在对 UPDATE 或 DELETE 语句使用 WHERE 子句前，应该先用 SELECT 进行测试，保证它过滤的是正确的记录，以防编写的 WHERE 子句不正确
- 使用强制实施引用完整性的数据库(关于这个内容，请参阅第 15 章)，这样 MySQL 将不允许删除具有与其他表相关联的数据的行

3、小心使用：MySQL 没有撤销(undo)按钮。应该非常小心地使用 UPDATE 和 DELETE，否则你会发现自己更新或删除了错误的数据

Chapter 21. 创建和操纵表

21.1. 创建表

1、MySQL 不仅用于表数据操纵，而且还可以用来执行数据库和表的所有操作，包括表本身的创建和处理

2、一般有两种创建表的方法

- 使用具有交互式创建和管理表的工具
- 表也可以直接用 MySQL 语句操纵

3、为了用程序创建表，可使用 SQL 的 CREATE TABLE 语句。值得注意的是，在使用交互式工具时，实际上使用的是 MySQL 语句。但是，这些语句不是用户编写的，界面工具会自动生成并执行相应的 MySQL 语句(更改现有表时也是样)。

21.1.1. 创建表的基础

1、为利用 CREATE TABLE 创建表，必须给出下列信息

- 新表的名字，在关键字 CREATE TABLE 之后给出
- 表列的名字和定义，用逗号分隔
- 每列的定义以列名(它在表中必须是唯一的)开始，后跟列的数据类型
- 表的主键可以在创建表时用 PRIMARY KEY 关键字指定

2、在创建新表时，指定的表名必须不存在，否则将出错。如果要防止意外覆盖已有的表，SQL 要求首先手工删除该表，然后再重建它，而不是简单地用创建表语句覆盖它。

3、如果你仅想在一个表不存在时创建它，应该在表名后给出 IF NOT EXISTS。这样做不检查已有表的模式是否与你打算创建的表模式相匹配。它只是查看表名是否存在，并且仅在表名不存在时创建它

21.1.2. 使用 NULL 值

1、允许 NULL 值的列也允许在插入行时不给出该列的值；不允许 NULL 值的列不接受该列没有值的行，换句话说，在插入或更新行时，该列必须有值

```
CREATE TABLE customers
(
   cust_id      int       NOT NULL AUTO_INCREMENT,
   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,
   PRIMARY KEY (cust_id)
)ENGING=InnoDB;
```

2、如果不指定 NOT NULL，则认为指定的是 NULL，因为 NULL 是默认值

3、理解 NULL：不要把 NULL 值与空串相混淆。NULL 值是没有值，它不是空串。如果指定''(两个单引号，其间没有字符)，这在 NOT NULL 列中是允许的。空串是一个有效的值，它不是无值。NULL 值用关键字 NULL 而不是空串指定

21.1.3. 主键再介绍

1、主键值必须唯一。即，表中的每个行必须具有唯一的主键值。如果主键使用单个列，则它的值必须唯一。如果使用多个列，则这些列的组合值必须唯一。

```
CREATE TABLE orderitems
(
  order_num    int        NOT NULL,
  order_item   int        NOT NULL,
  prod_id      char(10)   NOT NULL,
  quantity     int        NOT NULL,
  item_price   decimal(8,2)  NOT NULL,
  PRIMARY KEY (order_num,order_item)
)ENGINE=InnoDB;
```

2、主键可以在创建表时定义，或者在创建表之后定义

3、主键和 NULL 值：主键为其值唯一标识表中每个行的列

- **主键中只能使用不允许 NULL 值的列**
- **允许 NULL 值的列不能作为唯一标识**

21.1.4. 使用 AUTO_INCREMENT

1、AUTO_INCREMENT 告诉 MySQL，本列每当增加一行时自动增量。每次执行一个 INSERT 操作时，MySQL 自动对该列增量(从而才有这个关键字 AUTO_INCREMENT)，给该列赋予下一个可用的值

2、覆盖 AUTO_INCREMENT：如果一个列被指定为 AUTO_INCREMENT，则它需要使用特殊的值吗

- 你可以简单地在 INSERT 语句中指定一个值，只要它是唯一的(至今尚未使用过)即可，该值将被用来替代自动生成的值
- 后续的增量将开始使用该手工插入的值

21.1.5. 指定默认值

1、如果在插入行时没有给出值，MySQL 允许指定此时使用的默认值。默认值用 CREATE TABLE 语句的列定义中的 DEFAULT 关键字指定

2、不允许函数：与大多数 DBMS 不一样，MySQL 不允许使用函数作为默认值，它只支持常量

3、使用默认值而不是 NULL 值：许多数据库开发人员使用默认值而不是 NULL 列，特别是对用于计算或数据分组的列更是如此

```
CREATE TABLE orderitems
(
    order_item int          NOT NULL,
    prod_id      char(10)     NOT NULL,
    quantity     int        NOT NULL   DEFAULT 1,
    item_price     decimal(8,2)  NOT NULL,
    PRIMARY KEY(order_num,order_item)
) ENGINE=InnoDB;
```

21.1.6. 引擎类型

1、迄今为止使用的 CREATE TABLE 语句全都以 ENGINE=InnoDB 语句结束

2、与其他 DBMS 一样，MySQL 有一个具体管理和处理数据的内部引擎。在你使用 CREATE TABLE 语句时，该引擎具体创建表，而在你使用 SELECT 语句或进行其他数据库处理时，该引擎在内部处理你的请求。多数时候，此引擎都隐藏在 DBMS 内，不需要过多关注它

3、但 MySQL 与其他 DBMS 不一样，它具有多种引擎。它打包多个引擎，这些引擎都隐藏在 MySQL 服务器内，全都能执行 CREATE TABLE 和 SELECT 等命令。

4、为什么要发行多种引擎呢?因为它们具有各自不同的功能和特性，为不同的任务选择正确的引擎能获得良好的功能和灵活性。

5、当然，你完全可以忽略这些数据库引擎。如果省略 ENGINE=语句，则使用默认引擎(很可能是 MyISAM)，多数 SQL 语句都会默认使用它。但并不是所有语句都默认使用它，这就是为什么 ENGINE=语句很重要的原因

6、需要知道的引擎：

- InnoDB 是一个可靠的事务处理引擎(参见第 26 章)，它不支持全文本搜索
- MEMORY 在功能等同于 MyISAM，但由于数据存储在内存(不是磁盘)中，速度很快(特别适合于临时表)
- MyISAM 是一个性能极高的引擎，它支持全文本搜索(参见第 18 章)，但不支持事务处理。

7、外键不能跨引擎

- 混用引擎类型有一个大缺陷。外键(用于强制实施引用完整性，如第 1 章所述)不能跨引擎，即使用一个引擎的表不能引用具有使用不同引擎的表的外键

21. 2. 更新表

1、为更新表定义，可使用 ALTER TABLE 语句。但是，理想状态下，当表中存储数据以后，该表就不应该再被更新。在表的设计过程中需要花费大量时间来考虑，以便后期不对该表进行大的改动

2、为了使用 ALTER TABLE 更改表结构，必须给出下面的信息

- 在 ALTER TABLE 之后给出要更改的表名(该表必须存在，否则将出错)
- 所作更改的列表

3、例子

```
ALTER TABLE vendors
ADD vend_phone CHAR(20);

ALTER TABLE vendors
DROP COLUMN vend_phone
```

4、复杂的表结构更改一般需要手动删除过程，它涉及以下步骤

- 用新的列布局创建一个新表
- 使用 INSERT SELECT 语句从旧表复制数据到新表。如果有必要，可使用转换函数和计算字段
- 检验包含所需数据的新表
- 重命名旧表(如果确定，可以删除它)
- 用旧表原来的名字重命名新表
- 根据需要，重新创建触发器、存储过程、索引和外键

5、小心使用 ALTER TABLE：

- 使用 ALTER TABLE 要极为小心，应该在进行改动前做一个完整的备份(模式和数据的备份)
- 数据库表的更改不能撤销，如果增加了不需要的列，可能不能删除它们
- 类似地，如果删除了不应该删除的列，可能会丢失 该列中的所有数据

21.3. 删除表

1、删除表(删除整个表而不是其内容)非常简单，使用 DROP TABLE 语句即可

```
DROP TABLE customers;
```

21.4. 重命名表

2、使用 RENAME TABLE 语句可以重命名一个表

```
RENAME TABLE customers2 TO customers;
```

Chapter 22. 使用视图

22.1. 视图

1、需要 MySQL 5：MySQL 5 添加了对视图的支持。因此，本章内容适用于 MySQL 5 及以后的版本

2、视图是虚拟的表，与包含数据的表不同，视图只包含使用时动态检索数据的查询

3、用一个例子来理解

```
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 = 'TNT2'
```

- 假如可以把整个查询包装成一个名为 productcustomers 的虚拟表

```
SELECT cust_name, cust_contact
FROM productcustomers
WHERE prod_id = 'TNT2';
```

- productcustomers 是一个视图，作为视图，它不包含表中应该由的任何列或数据，它包含的是一个 SQL 查询

22.1.1. 为什么使用视图

1、视图的常见应用

- 重用 SQL 语句
- 简化复杂的 SQL 操作，在编写查询后，可以方便地重用它而不必知道它的基本查询细节
- 使用表的组成部分而不是整个表
- 保护数据，可以给用户授予表的特定部分的访问权限而不是整个表的访问权限
- 更改数据格式和表示，视图可返回与底层表的表示和格式不同的数据

2、在视图创建之后，可以用与表基本相同的方式利用它们。可以对视图执行 SELECT 操作，过滤和排序数据，将视图联结到其他视图或表，甚至能添加和更新数据

3、重要的是知道视图仅仅是用来查看存储在别处的数据的一种设施

- 视图本身不包含数据，因此它们返回的数据是从其他表中检索出来的
- 在添加或更改这些表中的数据时，视图将返回改变过的数据

4、性能问题

- 因为视图不包含数据，所以每次使用视图时，都必须处理查询执行时所需的任一个检索
- 如果你用多个联结和过滤创建了复杂的视图或者嵌套了视图，可能会发现性能下降得很厉害
- 因此，在部署使用了大量视图的应用前，应该进行测试

22.1.2. 视图的规则和限制

1、下面是关于视图创建和使用的一些最常见的规则和限制

- 与表一样，视图必须唯一命名(不能给视图取与别的视图或表相同的名字)
- 对于可以创建的视图数目没有限制
- 为了创建视图，必须具有足够的访问权限。这些限制通常由数据库管理人员授予
- 视图可以嵌套，即可以利用从其他视图中检索数据的查询来构造一个视图
- ORDER BY 可以用在视图中，但如果从该视图检索数据 SELECT 中也含有 ORDER BY，那么该视图中的 ORDER BY 将被覆盖
- 视图不能索引，也不能有关联的触发器或默认值
- 视图可以和表一起使用。例如，编写一条联结表和视图的 SELECT 语句。

22. 2. 使用视图

1、在理解什么是视图(以及管理它们的规则及约束)后，我们来看一下视图的创建

- 视图用 CREATE VIEW 语句来创建
- 使用 SHOW CREATE VIEW viewname;来查看创建视图的语句
- 用 DROP 删除视图，其语法为 DROP VIEW viewname
- 更新视图时，可以先用 DROP 再用 CREATE，也可以直接用 CREATE OR REPLACE VIEW
  - 如果要更新的视图不存在，则第 2 条更新语句会创建一个视图
  - 如果要更新的视图存在，则第 2 条更新语句会替换原有视图。

22. 2. 1. 利用视图简化复杂的联结

1、视图的最常见的应用之一是隐藏复杂的 SQL，这通常都会涉及联结

```
CREATE VIEW productcustomers AS
SELECT cust_name, cust_contact, prod_id
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
  AND orderitems.order_num = orders.order_num;

SELECT cust_name, cust_conact
FROM productcustomers
WHERE prod_id = 'TNT2';
```

2、创建可重用的视图：创建不受特定数据限制的视图是一种好办法。例如，上面创建的视图返回生产所有产品的客户而不仅仅是生产 TNT2 的客户。扩展视图的范围不仅使得它能被重用，而且甚至更有用。这样做不需要创建和维护多个类似视图

22. 2. 2. 用视图重新格式化检索出的数据

1、例子

```
CREATE VIEW vendorlocations AS
SELECT Concat(RTrim(vend_name),'(',RTrim(vend_country),')') AS vend_title
```

```
FROM vendors
ORDER BY vend_name;

SELECT *
FROM vendorlocations;
```

22. 2. 3. 用视图过滤不想要的数据

1、视图对于应用普通的 WHERE 子句也很有用

```
CREATE VIEW customermaillist AS
SELECT cust_id, cust_name, cust_email
FROM customers
WHERE cust_email IS NOT NULL;

SELECT *
FROM customermaillist;
```

22. 2. 4. 使用视图与计算字段

1、视图对于简化计算字段的使用特别有用

```
CREATE VIEW orderitemsexpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM orderitems;

SELECT *
FROM orderitemexpanded
WHERE order_num = 20005;
```

22. 2. 5. 更新视图

1、通常，视图是可更新的(即，可以对它们使用 INSERT、UPDATE 和 DELETE)。更新一个视图将更新其基表(可以回忆一下，视图本身没有数据)。如果你对视图增加或删除行，实际上是对其基表增加或删除行

2、但是，并非所有视图都是可更新的。基本上可以说，如果 MySQL 能正确地确定被更新的基数据，则不允许更新(包括插入和删除)。这实际上意味着，如果视图定义中有以下操作，则不能进行视图的更新

- 分组(使用 GROUP BY 和 HAVING)
- 联结
- 子查询
- 并
- 聚集函数(Min()、Count()、Sum()等)
- DISTINCT
- 导出(计算)列

Chapter 23. 使用存储过程

23.1. 存储过程

1、MySQL 5 添加了对存储过程的支持，因此，本章内容适用于 MySQL 5 及以后的版本

2、迄今为止，使用的大多数 SQL 语句都是针对一个或多个表的单条语句。并非所有操作都这么简单，经常会有一个完整的操作需要多条语句才能完成。例如，考虑如下情形

- 为了处理订单，需要核对以保证库存中有相应的物品
- 如果库存有物品，这些物品需要预定以便不将它们再卖给别的人，并且要减少可用的物品数量以反映正确的库存量
- 库存中没有的物品需要订购，这需要与供应商进行某种交互
- 关于哪些物品入库(并且可以立即发货)和哪些物品退订，需要通知相应的客户

3、可以创建存储过程。存储过程简单来说，就是为以后的使用而保存的一条或多条 MySQL 语句的集合。可将其视为批文件，虽然它们的作用不仅限于批处理

23.2. 为什么要使用存储过程

1、以下是一些主要的理由

- 通过把处理封装在容易使用的单元中，简化复杂的操作
- 由于不要求反复建立一系列处理步骤，这保证了数据的完整性。如果所有开发人员和应用程序都使用同一(试验和测试)存储过程，则所使用的代码都是相同的。这一点的延伸就是防止错误。需要执行的步骤越多，出错的可能性就越大。防止错误保证了数据的一致性
- 简化对变动的管理。如果表名、列名或业务逻辑(或别的内容)有变化，只需要更改存储过程的代码。使用它的人员甚至不需要知道这些变化
  - 这一点的延伸就是安全性。通过存储过程限制对基础数据的访问减少了数据讹误(无意识的或别的原因所导致的数据讹误)的机会
- 提高性能。因为使用存储过程比使用单独的 SQL 语句要快。
- 存在一些只能用在单个请求中的 MySQL 元素和特性，存储过程可 以使用它们来编写功能更强更灵活的代码
  - 换句话说，使用存储过程有 3 个主要的好处，即简单、安全、高性能。显然，它们都很重要。不过，在将 SQL 代码转换为存储过程前，也必须知 道它的一些缺陷。
- 一般来说，存储过程的编写比基本 SQL 语句复杂，编写存储过程需要更高的技能，更丰富的经验
- 你可能没有创建存储过程的安全访问权限。许多数据库管理员限制存储过程的创建权限，允许用户使用存储过程，但不允许他们创建存储过程

2、MySQL 将编写存储过 程的安全和访问与执行存储过程的安全和访问区分开来。这是好事情。即使你不能(或不想)编写自己的存储过程，也仍然可以在适当的时候执行别的存储过程

23.3. 使用存储过程

1、使用存储过程需要知道如何执行(运行)它们。存储过程的执行远比其定义更经常遇到，因此，我们将从执行存储过程开始介绍。然后再介绍创建和使用存储过程

23.3.1. 执行存储过程

1、MySQL 称存储过程的执行为调用，因此 MySQL 执行存储过程的语句为 CALL。CALL 接受存储过程的名字以及需要传递给它的任意参数

23.3.2. 创建存储过程

1、例子

```
CREATE PROCEDURE productpricing()
BEGIN
    SELECT Avg(prod_price) AS priceaverage
    FROM products;
END;
```

- 如果存储过程接受参数，它们将在()中列举出来，若没有参数，()仍然需要
- BEGIN 和 END 语句用来限定存储过程体

2、默认的 MySQL 语句分隔符为;(正如你已经在迄今为止所使用的 MySQL 语句中所看到的那样)。mysql 命令行实用程序也使用;作为语句分隔符。如果命令行实用程序要解释存储过程自身内的;字符，则它们最终不会成为存储过程的成分，这会使存储过程中的 SQL 出现句法错误。

- 解决办法是临时更改命令行实用程序的语句分隔符，如下所示

```
DELIMITER //

CREATE PROCEDURE productpricing()
BEGIN
    SELECT Avg(prod_price) AS priceaverage
    FROM products;
END //
```

- DELIMITER //告诉命令行实用程序使用//作为新的语句结束分隔符，可以看到标志存储过程结束的 END 定义为 END//而不是 END;
- 这样，存储过程体内的;仍然保持不动，并且正确地传递给数据库引擎。最后，为恢复为原来的语句分隔符
- 除\符号外，任何字符都可以用作语句分隔符

3、使用

```
CALL productpricing();
```

23.3.3. 删除存储过程

1、储过程在创建之后，被保存在服务器上以供使用，直至被删除。删除命令(类似于第 21 章所介绍的语句)从服务器中删除存储过程

```
DROP PROCEDURE productpricing;
```

2、仅当存在时删除

- 如果指定的过程不存在，则 DROPPROCEDURE 将产生一个错误
- 当过程存在想删除它时(如果过程不存在也 不产生错误)可使用 DROP PROCEDURE IF EXISTS。

23. 3. 4. 使用参数

1、一般，存储过程并不显示结果，而是把结果返回给你指定的变量

- 变量：内存中一个特定的位置，用来临时存储数据

```
CREATE PROCEDURE productpricing(
    OUT pl DECIMAL(8,2)
    OUT ph DECIMAL(8,2)
    OUT pa DECIMAL(8,2)
)
BEGIN
    SELECT Min(Prod_price)
    INTO pl
    FROM products;
    SELECT Max(prod_price)
    INTO ph
    FROM products;
    SELECT Avg(prod_price)
    INTO pa
    FROM products;
END
```

- MySQL 支持 IN(传递给存储过程)、OUT(从存储过程传出，如这里所用)和 INOUT(对存储过程传入和传出)类型的参数
- INTO：用于将查询结果保存到指定变量

2、调用

```
CALL productpricing(@pricelow,@pricehigh,@priceaverage);
```

- 所有 MySQL 变量都必须以@开始

3、查看结果值

```
SELECT @pricehigh,@pricelow,@priceaverage;
```

4、另一个例子

```
CREATE PROCEDURE ordertotal(
    IN onumber INT,
    OUT ototal DECIMAL(8,2)
)
BEGIN
    SELECT Sum(item_price*quantity)
    FROM orderitems
    WHERE order_num=onumber
    INTO ototal;
END;

CALL ordertotal(20005,@total);

SELECT @total;
```

23.3.5. 建立智能存储过程

1、迄今为止使用的所有存储过程基本上都是封装 MySQL 简单的 SELECT 语句。虽然它们全都是有效的存储过程例子，但它们所能完成的工作你直接用这些被封装的语句就能完成(如果说它们还能带来更多的东西那就是使事情更复杂)。只有在存储过程内包含业务规则和智能处理时，它们的威力才真正显现出来

23.3.6. 检查存储过程

1、为显示用来创建一个存储过程的 CREATE 语句，使用 SHOW CREATE PROCEDURE 语句

```
SHOW CREATE PROCEDURE ordertotal;
```

23.4. 小结

1、一句话：存储过程说白了就是函数

Chapter 24. 使用游标

24.1. 游标

1、MySQL 检索操作返回一组称为结果集的行。这组返回的行都是与 SQL 语句相匹配的行(零行或多行)。使用简单的 SELECT 语句，例如，没有办法得到第一行、下一行或前 10 行，也不存在每次一行地处理所有行的简单方法(相对于成批地处理它们)

2、有时，需要在检索出来的行中前进或后退一行或多行。这就是使用游标的原因。游标(cursor)是一个存储在 MySQL 服务器上的数据库查询， 它不是一条 SELECT 语句，而是被该语句检索出来的结果集。在存储了游标之后，应用程序可以根据需要滚动或浏览其中的数据

3、游标主要用于交互式应用，其中用户需要滚动屏幕上的数据，并对数据进行浏览或做出更改

24.2. 使用游标

1、使用游标涉及几个明确的步骤

1) 在能够使用游标前，必须声明(定义)它。这个过程实际上没有检索数据，它只是定义要使用的 SELECT 语句
2) 一旦声明后，必须打开游标以供使用。这个过程用前面定义的 SELECT 语句把数据实际检索出来
3) 对于填有数据的游标，根据需要取出(检索)各行
4) 在结束游标使用时，必须关闭游标

24.2.1. 创建游标

1、游标用 DECLARE 语句创建(参见第 23 章)。DECLARE 命名游标，并定义相应的 SELECT 语句，根据需要带 WHERE 和其他子句。例如，下面的语句定义了名为 ordernumbers 的游标，使用了可以检索所有订单的 SELECT 语句

```
CREATE PROCEDURE processors()
BEGIN
    DECLARE ordernumbers CURSOR
    FOR
    SELECT order_num FROM orders;
END;
```

- 游标定义放在了存储过程中，但其实游标与存储过程并无必然关系，只是这个例子将游标放在存储过程中而已

24.2.2. 打开和关闭游标

1、游标用 OPEN CURSOR 语句来打开

```
OPEN ordernumbers;
```

2、游标处理完成后，应当使用如下语句关闭游标

```
CLOSE ordernumbers;
```

3、在一个游标关闭后，如果没有重新打开，则不能使用它。但是，使 用声明过的游标不需要再次声明，用 OPEN 语句打开它就可以了

```
CREATE PROCEDURE processors()
BEGIN
```

```
    DECLARE ordernumbers CURSOR
    FOR
    SELECT order_num FROM orders;

    OPEN ordernums;

    CLOSE ordernumbers;
END;
```

24. 2. 3. 使用游标数据

1、在一个游标被打开后，可以使用 FETCH 语句分别访问它的每一行。FETCH 指定检索什么数据(所需的列)，检索出来的数据存储在什么地方。它还向前移动游标中的内部行指针，使下一条 FETCH 语句检索下一行(不重复读取同一行)

```
CREATE PROCEDURE processors()
BEGIN

    DECLARE o INT;

    DECLARE ordernumbers CURSOR
    FOR
    SELECT order_num FROM orders;

    OPEN ordernums;
    FETCH ordernumbers INTO o;

    CLOSE ordernumbers;
END;


CREATE PROCEDURE processors()
BEGIN

    DECLARE done BOOLEAN DEFAULT 0;
    DECLARE o INT;

    DECLARE ordernumbers CURSOR
    FOR
    SELECT order_num FROM orders;

    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done=1;

    OPEN ordernums;
    REPEAT
        FETCH ordernumbers INTO o;
    UNTIL done END REPEAT;

    CLOSE ordernumbers;
```

```
END;
```

Chapter 25. 使用触发器

25.1. 触发器

1、MySQL 语句在需要时被执行，存储过程也是如此。但是，如果你想要某条语句(或某些语句)在事件发生时自动执行，怎么办呢?例如

1) 每当增加一个顾客到某个数据库表时，都检查其电话号码格式是否正确，州的缩写是否为大写
2) 每当订购一个产品时，都从库存数量中减去订购的数量
3) 无论何时删除一行，都在某个存档表中保留一个副本

2、触发器是 MySQL 响应以下任意语句而自动执行的一条 MySQL 语句(或位于 BEGIN 和 END 语句之间的一组语句)

1) DELETE
2) INSERT
3) UPDATE

25.2. 创建触发器

1、在创建触发器时，需要给出 4 条信息:

1) 唯一的触发器名
2) 触发器关联的表
3) 触发器应该响应的活动(DELETE、INSERT 或 UPDATE)
4) 触发器何时执行(处理之前或之后)

2、保持每个数据库的触发器名唯一：在 MySQL5 中，触发器名必须在每个表中唯一，但不是在每个数据库中唯一。这表示同一数据库中的两个表可具有相同名字的触发器。这在其他每个数据库触发器名必须唯一的 DBMS 中是不允许的，而且以后的 MySQL 版本很可能会使命名规则更为严格。因此，现在最好是在数据库范围内使用唯一的触发器名

3、触发器用 CREATE TRIGGER 语句创建

```
CREATE TRIGGER newproduct AFTER INSERT ON products
FOR EACH ROW SELECT 'Product added';
```

4、触发器按每个表每个事件每次地定义，每个表每个事件每次只允许一个触发器。因此，每个表最多支持 6 个触发器(每条 INSERT、UPDATE 和 DELETE 的之前和之后)。单一触发器不能与多个事件或多个表关联，所以，如果你需要一个对 INSERT 和 UPDATE 操作执行的触发器，则应该定义两个触发器

25.3. 删除触发器

1、为了删除一个触发器，可使用 DROP TRIGGER 语句

```
DROP TRIGGER newproduct;
```

2、触发器不能更新或覆盖。为了修改一个触发器，必须先删除它， 然后再重新创建

25.4. 使用触发器

25.4.1. INSERT 触发器

1、INSERT 触发器在 INSERT 语句执行之前或之后执行。需要知道以下几点

1) 在 INSERT 触发器代码内，可引用一个名为 NEW 的虚拟表，访问被插入的行
2) 在 BEFORE INSERT 触发器中，NEW 中的值也可以被更新(允许更改被插入的值)
3) 对于 AUTO_INCREMENT 列，NEW 在 INSERT 执行之前包含 0，在 INSERT 执行之后包含新的自动生成值

2、例子

```
CREATE TRIGGER neworder AFTER INSERT ON orders
FOR EACH ROW SELECT NEW.order_num;
```

25.4.2. DELETE 触发器

1、DELETE 触发器在 DELETE 语句执行之前或之后执行。需要知道以下两点

1) 在 DELETE 触发器代码内，你可以引用一个名为 OLD 的虚拟表，访问被删除的行
2) OLD 中的值全都是只读的，不能更新

2、下面的例子演示使用 OLD 保存将要被删除的行到一个存档表中

```
DELEMITER $$
CREATE TRIGGER deleteorder BEFORE DELETE ON orders
FOR EACH ROW
BEGIN
    INSERT INTO archive_orders(order_num,order_date,cust_id)
    VALUES(OLD.order_num,OLD.order_date,OLD.cust_id);
END $$
```

25.4.3. UPDATE 触发器

1、UPDATE 触发器在 UPDATE 语句执行之前或之后执行。需要知道以下几点

1) 在 UPDATE 触发器代码中，你可以引用一个名为 OLD 的虚拟表访问以前(UPDATE 语句前)的值，引用一个名为 NEW 的虚拟表访问新更新的值
2) 在 BEFORE UPDATE 触发器中，NEW 中的值可能也被更新(允许更改将要用于 UPDATE 语句中的值)
3) OLD 中的值全都是只读的，不能更新

2、下面的例子保证州名缩写总是大写(不管 UPDATE 语句中给出的是大写还是小写)

```
CREATE TRIGGER updatevendor BEFORE UPDATE ON vendors
FOR EACH ROW SET NEW.vend_state=Upper(New.vend_state);
```

25.4.4. 关于触发器的进一步介绍

1、与其他 DBMS 相比，MySQL 5 中支持的触发器相当初级。未来的 MySQL 版本中有一些改进和增强触发器支持的计划

2、创建触发器可能需要特殊的安全访问权限，但是，触发器的执行是自动的。如果 INSERT、UPDATE 或 DELETE 语句能够执行，则相关 的触发器也能执行

3、应该用触发器来保证数据的一致性(大小写、格式等)。在触发器中执行这种

类型的处理的优点是它总是进行这种处理，而且是透明地进行，与客户机应用无关

4、触发器的一种非常有意义的使用是创建审计跟踪。使用触发器，把更改(如果需要，甚至还有之前和之后的状态)记录到另一个表非常容易

5、遗憾的是，MySQL 触发器中不支持 CALL 语句。这表示不能从触发器内调用存储过程。所需的存储过程代码需要复制到触发器内

Chapter 26. 管理事务处理

26.1. 事务处理

1、并非所有引擎都支持事务处理：正如第 21 章所述，MySQL 支持几种基本的数据库引擎。正如本章所述，并非所有引擎都支持明确的事务处理管理。MyISAM 和 InnoDB 是两种最常使用的引擎。前者不支持明确的事务处理管理，而后者支持。这就是为什么本书中使用的样例表被创建来使用 InnoDB 而不是更经常使用的 MyISAM 的原因。如果你的应用中需要事务处理功能，则一定要使用正确的引擎类型

2、事务处理(transaction processing)可以用来维护数据库的完整性，它保证成批的 MySQL 操作要么完全执行，要么完全不执行

3、事务处理是一种机制，用来管理必须成批执行的 MySQL 操作，以保证数据库不包含不完整的操作结果。利用事务处理，可以保证一组操作不会中途停止，它们或者作为整体执行，或者完全不执行(除非明确指示)。如果没有错误发生，整组语句提交给(写到)数据库表。如果发生错误，则进行回退(撤销)以恢复数据库到某个已知且安全的状态

4、下面是关于 事务处理需要知道的几个术语

1) 事务(transaction)指一组 SQL 语句
2) 回退(rollback)指撤销指定 SQL 语句的过程
3) 提交(commit)指将未存储的 SQL 语句结果写入数据库表
4) 保留点(savepoint)指事务处理中设置的临时占位符(place-holder)，你可以对它发布回退(与回退整个事务处理不同)。

26.2. 控制事务处理

1、管理事务处理的关键在于将 SQL 语句组分解为逻辑块，并明确规定数据何时应该回退，何时不应该回退

2、MySQL 使用下面的语句来标识事务的开始

```
START TRANSACTION
```

26.2.1. 使用 ROLLBACK

1、MySQL 的 ROLLBACK 命令用来回退(撤销)MySQL 语句，请看下面的语句

```
SELECT * FROM ordertotals;
START TRANSACTION;
DELETE FROM ordertotals;
SELECT *FROM ordertotals;
ROLLBACK;
SELECT * FROM ordertotals;
```

2、哪些语句可以回退?事务处理用来管理 INSERT、UPDATE 和 DELETE 语句。你不能回退 SELECT 语句。(这样做也没有什么意义。)你不能回退 CREATE 或 DROP 操作。事务处理块中可以使用这两条语句，但如果你执行回退，它们不会被撤销

26.2.2. 使用 COMMIT

1、一般的 MySQL 语句都是直接针对数据库表执行和编写的。这就是所谓的隐

含提交(implicit commit)，即提交(写或保存)操作是自动进行的
2、但是，在事务处理块中，提交不会隐含地进行。为进行明确的提交，使用 COMMIT 语句
3、隐含事务关闭：当 COMMIT 或 ROLLBACK 语句执行后，事务会自动关闭(将来的更改会隐含提交)

26. 2. 3. 使用保留点

1、简单的 ROLLBACK 和 COMMIT 语句就可以写入或撤销整个事务处理。但是，只是对简单的事务处理才能这样做，更复杂的事务处理可能需要部分提交或回退
2、为了支持回退部分事务处理，必须能在事务处理块中合适的位置放置占位符。这样，如果需要回退，可以回退到某个占位符
3、这些占位符称为保留点。为了创建占位符，可如下使用 SAVEPOINT 语句

```
SAVEPOINT delete1;
```

4、每个保留点都取标识它的唯一名字，以便在回退时，MySQL 知道要 回退到何处。为了回退到本例给出的保留点，可如下进行

```
ROLLBACK TO delete1;
```

5、释放保留点：保留点在事务处理完成(执行一条 ROLLBACK 或 COMMIT)后自动释放。自 MySQL 5 以来，也可以用 RELEASE SAVEPOINT 明确地释放保留点

26. 2. 4. 更改默认的提交行为

1、正如所述，默认的 MySQL 行为是自动提交所有更改。换句话说，任何时候你执行一条 MySQL 语句，该语句实际上都是针对表执行的，而且所做的更改立即生效。为指示 MySQL 不自动提交更改，需要使用以下语句

```
SET autocommit=0;
```

2、标志为连接专用：autocommit 标志是针对每个连接而不是服务器的

Chapter 27. 全球化和本地化

27.1. 字符集合校对顺序

1、数据库表被用来存储和检索数据。不同的语言和字符集需要以不同的方式存储和检索。因此，MySQL 需要适应不同的字符集(不同的字母和字符)，适应不同的排序和检索数据的方法

2、术语

1) 字符集：字母和符号的集合
2) 编码：某个字符集成员的内部表示
3) 校对：规定字符如何比较的指令

27.2. 使用字符集合校对顺序

1、MySQL 支持众多的字符集。为查看所支持的字符集完整列表，使用以下语句

```
SHOW CHARACTER SET;
```

2、为了查看所支持校对的完整列表，使用以下语句

```
SHOW COLLATION;
```

3、通常系统管理在安装时定义一个默认的字符集和校对。此外，也可以在创建数据库时，指定默认的字符集和校对。为了确定所用的字符集和校对，可以使用以下语句

```
SHOW VARIABLES LIKE 'character%';
SHOW VARIABLES LIKE 'collation%';
```

4、为了给表指定字符集和校对，可使用带子句的 CREATE TABLE

```
CREATE TABLE mytable(
   column1 INT,
   column2 VARCHAR(10)
)DEFAULT CHARACTER SET hebrew
COLLATE hebrew_general_ci;
```

- 这个例子中指定了 CHARACTER SET 和 COLLATE 两者。一般，MySQL 如下确定使用什么样的字符集和校对

1) 如果指定 CHARACTER SET 和 COLLATE 两者，则使用这些值
2) 如果只指定 CHARACTER SET，则使用此字符集及其默认的校对(如 SHOW CHARACTER SET 的结果中所示)
3) 如果既不指定 CHARACTER SET，也不指定 COLLATE，则使用数据库默认

Chapter 28. 安全管理

28.1. 访问控制

1、MySQL 服务器的安全基础是：用户应该对他们需要的数据具有适当的访问权，既不能多也不能少。换句话说，用户不能对过多的数据具有过多的访问权

2、考虑以下内容

1) 多数用户只需要对表进行读和写，但少数用户甚至需要能创建和删除表
2) 某些用户需要读表，但可能不需要更新表
3) 你可能想允许用户添加数据，但不允许他们删除数据
4) 某些用户(管理员)可能需要处理用户账号的权限，但多数用户不需要
5) 你可能想让用户通过存储过程访问数据，但不允许他们直接访问数据
6) 你可能想根据用户登录的地点限制对某些功能的访问

28.2. 管理用户

1、MySQL 用户账号和信息存储在名为 mysql 的 MySQL 数据库中。一般不需要直接访问 mysql 数据库和表(你稍后会明白这一点)，但有时需要直接访问。需要直接访问它的时机之一是在需要获得所有用户账号列表时。为此，可使用以下代码

```
USE mysql
SELECT user FROM user;
```

28.2.1. 创建用户账号

1、为了创建一个新用户账号，使用 CREATE USER 语句

```
CREATE USER ben IDENTIFIED BY 'p@$$wOrd'
```

2、指定散列口令：IDENTIFIEDBY 指定的口令为纯文本，MySQL 将在保存到 user 表之前对其进行加密。为了作为散列值指定口令，使用 IDENTIFIED BY PASSWORD

3、使用 GRANT 或 INSERT：GRANT 语句(稍后介绍)也可以创建用户账号，但一般来说 CREATE USER 是最清楚和最简单的句子。此外，也可以通过直接插入行到 user 表来增加用户，不过为安全起见，一般不建议这样做。MySQL 用来存储用户账号信息的表(以及表模式等)极为重要，对它们的任何毁坏都可能严重地伤害到 MySQL 服务器。因此，相对于直接处理来说，最好是用标记和函数来处理这些表

4、为重新命名一个用户账号，使用 RENAME USER 语句，如下所示

```
RENAME USER ben TO bforta;
```

28.2.2. 删除用户账号

1、为了删除一个用户账号(以及相关的权限)，使用 DROP USER 语句，如下所示

```
DROP USER bforta;
```

28.2.3. 设置访问权限

1、在创建用户账号后，必须接着分配访问权限。新创建的用户账号没有访问权限。它们能登录 MySQL，但不能看到数据，不能执行任何数据库操作

2、为看到赋予用户账号的权限，使用 SHOW GRANTS FOR

SHOW GRANTS FOR bforta;

3、为设置权限，使用 GRANT 语句。GRANT 要求你至少给出以下信息：

1) 要授予的权限
2) 被授予访问权限的数据库或表
3) 用户名

4、示例

GRANT SELECT ON crashcourse.* TO bforta;

1) 此 GRANT 允许用户在 crashcourse.*(crashcourse 数据库的所有表)上使用 SELECT
2) 通过只授予 SELECT 访问权限，用户 bforta 对 crashcourse 数据库中的所有数据具有只读访问权限

5、GRANT 的反操作为 REVOKE，用它来撤销特定的权限

REVOKE SELECT ON crashcourse.* FROM bforta;

6、GRANT 和 REVOKE 可在几个层次上控制访问权限

1) 整个服务器，使用 GRANT ALL 和 REVOKE ALL
2) 整个数据库，使用 ON database.*
3) 特定的表，使用 ON database.table
4) 特定的列
5) 特定的存储过程

7、所有权限

1) ALL：除 GRANT OPTION 外的所有权限
2) ALTER：使用 ALTER TABLE
3) ALTER ROUTINE：使用 ALTER PROCEDURE 和 DROP PROCEDURE
4) CREATE：使用 CREATE TABLE
5) CREATE ROUTINE：使用 CREATE PROCEDURE
6) CREATE TEMPORARY TABLES：使用 CREATE TEMPORARY TABLE
7) CREATE USER：使用 CREATE USER、DROP USER、RENAME USER 和 REVOKE ALL PRIVILEGES
8) CREATE VIEW：使用 CREATE VIEW
9) DELETE：使用 DELETE
10) DROP：使用 DROP TABLE
11) EXECUTE：使用 CALL 和存储过程
12) FILE：使用 SELECT INTO OUTFILE 和 LOAD DATA INFILE
13) GRANT OPTION：使用 GRANT 和 REVOKE
14) INDEX：使用 CREATE INDEX 和 DROP INDEX
15) INSERT：使用 INSERT
16) LOCK TABLES：使用 LOCK TABLES
17) PROCESS：使用 SHOW FULL PROCESSLIST
18) RELOAD：使用 FLUSH
19) REPLICATION CLIENT：服务器位置的访问
20) REPLICATION SLAVE：由复制从属使用
21) SELECT：使用 SELECT
22) SHOW DATABASES：使用 SHOWDATABASES

23) SHOW VIEW：使用 SHOWCREATEVIEW
24) SHUTDOWN：使用 mysqladmin shutdown(用来关闭 MySQL)
25) SUPER：使用 CHANGE MASTER、KILL、LOGS、PURGE、MASTER 和 SET GLOBAL。还允许 mysqladmin 调试登录
26) UPDATE：使用 UPDATE
27) USAGE：无访问权限

8、简化多次授权：可通过列出各权限并用逗号分隔，将多条 GRANT 语句串在一起，如下所示

```
GRANT SELECT,INSERT ON crashcourse.* TO bforta;
```

28. 2. 4. 更改口令

1、为了更改用户口令，可使用 SET PASSWORD 语句。新口令必须如下加密

```
SET PASSWORD FOR bforta=Password('n3w p@$$wOrd');
SET PASSWORD = Password('n3w p@$$wOrd');
```

28. 2. 5. 密码限制

1、详见 http://www.cnblogs.com/ivictor/p/5142809.html

2、以下命令会报错

```
CREATE USER hcf IDENTIFIED BY '123456';
```

ERROR 1819 (HY000): Your password does not satisfy the current policy requirements

3、这个其实与 validate_password_policy 的值有关

1) 0 or LOW：Length
2) 1 or MEDIUM：Length; numeric, lowercase/uppercase, and special characters
3) 2 or STRONG：Length; numeric, lowercase/uppercase, and special characters; dictionary file

4、默认是 1，即 MEDIUM，所以刚开始设置的密码必须符合长度，且必须含有数字，小写或大写字母，特殊字符

5、修改 validate_password_policy 参数的值

```
SET GLOBAL validate_password_policy=0;
SELECT @@validate_password_policy;
```

Chapter 29. 数据库维护

29.1. 数据备份

29.2. 进行数据库维护

29.3. 诊断启动问题

29.4. 查看日志文件

Chapter 30. 数据库的四大特性以及隔离级别

30.1. 四大特性

1、如果一个数据库声称支持事务的操作，那么该数据库必须要具备以下四个特性

1) 原子性
2) 一致性
3) 隔离性
4) 持久性

30.1.1. 原子性（Atomicity）

1、原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，这和前面两篇博客介绍事务的功能是一样的概念，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响

30.1.2. 一致性（Consistency）

1、一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态

2、拿转账来说，假设用户 A 和用户 B 两者的钱加起来一共是 5000，那么不管 A 和 B 之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是 5000，这就是事务的一致性

30.1.3. 隔离性（Isolation）

1、隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离

2、即要达到这么一种效果：对于任意两个并发的事务 T1 和 T2，在事务 T1 看来，T2 要么在 T1 开始之前就已经结束，要么在 T1 结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行

3、关于事务的隔离性数据库提供了多种隔离级别，稍后会介绍到

30.1.4. 持久性（Durability）

1、持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作

2、例如我们在使用 JDBC 操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误

30.2. 问题

1、以上介绍完事务的四大特性(简称 ACID)，现在重点来说明下事务的隔离性，当多个线程都开启事务操作数据库中的数据时，数据库系统要能进行隔离操作，以保证各个线程获取数据的准确性，在介绍数据库提供的各种隔离级别之前，

我们先看看如果不考虑事务的隔离性，会发生的几种问题

30.2.1. 脏读

1、脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据
2、当一个事务正在多次修改某个数据，而在这个事务中这多次的修改都还未提交，这时一个并发的事务来访问该数据，就会造成两个事务得到的数据不一致

30.2.2. 不可重复读

1、不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了
2、不可重复读和脏读的区别是，脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据
3、不可重复读的本质是：在读事务的过程中，有其他写事务的干扰，导致在同一个读事务中，两次读取的状态不同

30.2.3. 幻读

1、幻读是事务非独立执行时发生的一种现象。例如事务 T1 对一个表中所有的行的某个数据项做了从"1"修改为"2"的操作，这时事务 T2 又对这个表中插入了一行数据项，而这个数据项的数值还是为"1"并且提交给数据库。而操作事务 T1 的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务 T2 中添加的，就好像产生幻觉一样，这就是发生了幻读

30.3. 隔离性

1、MySQL 数据库为我们提供的四种隔离级别

1) Serializable (串行化)：可避免脏读、不可重复读、幻读的发生
2) Repeatable read (可重复读)：可避免脏读、不可重复读的发生
3) Read committed (读已提交)：可避免脏读的发生
4) Read uncommitted (读未提交)：最低级别，任何情况都无法保证

DESC 表名