## 阅读目标：

1. 快速学会并熟练使用SQL
2. 如何使用SQL开发应用程序
3. 在无人帮助的情况下有效而快速地使用SQL

## 数据库管理系统（DBMS）

1. Apache Open Office Base
2. IBM DB2
3. Microsoft Access
4. Microsoft SQL Server
5. MariaDB
6. MySQL
7. Oracle
8. PostgreSQL
9. SQLite

## 第1课 了解SQL

### 数据库基础

1. 数据库是一个以某种有组织的方式存储的数据集合
2. 数据库（database）：保存有组织的数据的容器
3. DBMS数据库管理系统：数据库软件应称为数据库管理系统。数据库是通过DBMS创建和操纵的容器，而具体它究竟是什么，形式如何，各种数据库都不一样。
4. 表是一种结构化的文件，可用来存储某种特定类型的数据
5. 表（table）：某种特定类型数据的结构化清单。关键点在于，存储在表中的数据是同一种类型的数据或清单。数据库中表的名字是唯一的，在不同的数据库中完全可以使用相同的表名。
6. 模式（schema）：关于数据库和表的布局及特性的信息。表具有一些特性，用来定义数据在表中如何存储、存储什么样的数据、数据如何分解，各部分信息如何命名等信息。模式可以用来描述数据库中特定的表，也可以用来描述整个数据库。
7. 列（column）：表中的一个字段。所有表都是由一个或多个列组成的。

8. 数据类型（datatype）：所允许的数据的类型。每个列表都有相应的数据类型，它限制该列中存储的数据。数据类型还帮助正确地分类数据，并在优化磁盘使用方面起重要作用。
9. 行（row）：表中的一个记录（record）
10. 主键（primary key）：一列（或一组列），其值能够唯一标识表中的每一行。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；每一行都必须具有一个主键值（主键列不允许NULL值）；主键列中的值不允许修改或更新；主键值不能重用（如果某行从表中删除，它的主键不能赋给以后的新行）

### 什么是SQL

1. SQL是一种专门用来与数据库沟通的语言，是Structured Query Language（结构化查询语言）的缩写。设计SQL的目的是为了提供一种从数据库中读写数据的简单有效的方法。
2. SQL的优点：几乎所有DBMS都支持SQL；SQL简单易学；SQL灵活易用，可进行非常复杂和高级的数据库操作。

### 小结：

- 什么是SQL
- SQL有什么用
- 基本的数据库术语

## 第2课 检索数据

如何使用SELECT语句从表中检索一个或多个数据列

### SELECT语句

1. 关键词（keyword）：作为SQL组成部分的保留字。关键字不能用作表或列的名字。
2. 每个SQL语句都是由一个或多个关键字构成的
3. 检索单个列：必须给出两个条件，想选择什么以及从哪选择。例如，利用SELECT语句从Products表中检索一个名为prod_name的列。返回的是表中所有行，未过滤也未排序的数据。
4. 多条SQL语句必须以分号分隔
5. SQL语句不区分大小写

6. 在处理SQL语句时，所有空格都被忽略
7. 检索多个列：列名之间必须以逗号分隔
8. 检索所有列：使用星号通配符；列的顺序一般是列在表定义中出现的物理顺序（数据返回给应用程序，根据需要进行格式化，再表示出来）
9. 检索不同的值：使用DISTINCT关键字，它指示数据库只返回不同的值；DISTINCT关键字作用于所有的列
10. 限制结果：在MySQL数据库中使用LIMIT关键字来限制返回行数；LIMIT 5 OFFSET 2指示MySQL等DBMS返回从第2行起的5行数据。第一个数字是检索的行数，第二个数字是指从哪里开始检索
11. 第0行：第一个被检索的行是第0行，而不是第1行。因此，LIMIT 1 OFFSET 1检索的是第2行，而不是第1行。
12. 使用注释：SQL语句是由DBMS处理的指令。添加注释使用两个连字符（——）嵌在行内或者使用/*之间都是注释*/

小结：

- 如何使用SQL的SELECT语句来检索单个列表、多个列表以及所有列表
- 如何返回不同的值
- 如何注释代码

## 第3课 排序检索数据

如何使用SELECT语句的ORDER BY子句，根据需要排序检索出的数据

### 排序数据

1. 子句（clause）：SQL语句由子句构成，有些子句是必需的，有些则是可选的。一个子句通常由一个关键字加上所提供的数据组成。
2. ORDER BY子句：取一个或多个子句，据此对SELECT语句检索出的数据进行排序
3. ORDER BY子句的位置：确保是SELECT语句中最后一条子句，不然会出现错误信息
4. ORDER BY子句可对非选择列进行排序
5. 按多个列排序：简单指定列名，列名之间用逗号分开
6. 按列位置排序：支持按SELECT清单中指定的选择列的相对位置排序，容易出错
7. 指定排序方向：升序ASC（ASCENDING） / 降序DESC（DESCENDING）
8. 在多个列上降序排序：关键字DESC只对指定列有效，想在多个列上降序排序，

必须对每一列指定DESC关键字

小结：

- 如何使用SQL的SELECT语句的ORDER BY子句对检索出的数据进行排序
- ORDER BY子句必须是SELECT语句中的最后一条子句
- 如何对一个或多个列进行升序 / 降序排序

## 第4课 过滤数据

如何使用SELECT语句的WHERE子句指定搜索条件

### 使用WHERE子句

1. 过滤条件（filter condition）：根据报告的需要指定搜索条件（search criteria）检索数据
2. SQL过滤与应用过滤：使用客户端过滤数据，需要服务器通过网络发送多余的数据，会造成网络带宽的浪费。优化数据库后可以快速有效地对数据进行过滤
3. WHERE子句的位置：ORDER BY子句应位于最后一句
4. WHERE子句操作符：等于（=） / 不等于（!= or <>） / 小于（<） / 小于等于（<=） / 不小于（!<） / 大于（>） / 大于等于（>=） / 不大于（!>） / 在指定的两个值之间（BETWEEN） / 为NULL值（IS NULL）
5. 检索单个值：

```
WHERE prod_price <= 10;
```

1. 不匹配检查：

```
WHERE vend_id <> 'DLL01';
```

1. 何时使用引号：值跟字符串类型的列进行比较，就需要限定引号；用来与数值列进行比较的值不用引号
2. 范围值检查：要检查某个范围的值，可以使用BETWEEN操作符，需要指定范围的开始值和结束值，这两个值必须使用AND关键字分隔。BETWEEN匹配范围中所有的值，包括指定的开始值和结束值。

```
WHERE prod_price BETWEEN 5 AND 10;
```

1. 空值检查：表设计人员在指定其中的列不包含值时，可以称其包含空值NULL。NULL无值（no value）与字段包含0、空字符串或仅仅包含空格不同。
2. 检索NULL值：SELECT 语句中有一个特殊的WHERE子句IS NULL，可以用来检索空值。

```
WHERE prod_price  IS NULL;
```

1. NULL和非匹配：在进行匹配过滤或非匹配过滤时，不会返回含NULL值的行；过滤数据时，一定要验证被过滤列中含NULL的行确实出现在返回的数据中

**小结：**

- 如何使用SQL的SELECT语句的WHERE子句过滤返回的数据
- 如何检验相等、不相等、大于、小于、值的范围以及NULL值

## 第5课 高级数据过滤

- 如何使用WHERE子句建立功能更强、更高级的搜索条件
- 如何使用NOT和IN操作符

### 组合WHERE子句

1. 操作符（operator）：用来联结或改变WHERE子句中的子句的关键字，也称为逻辑操作符（logical operator）AND/OR
2. AND操作符：通过AND操作符可以使用多个条件进行过滤

```
WHERE vend_id = 'DLL01' AND prod_price <= 4;
```

1. AND：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
2. OR：指示DBMS检索匹配任一条件的行
3. 求值顺序：WHERE子句可以包含任意数目的AND和OR操作符；允许两者结合以进行复杂、高级的过滤。AND在求值过程中优先级更高，需要使用圆括号对操作符进行明确分组。

```
WHERE ( vend_id = 'DLL01' OR vend_id = 'BRS01' ) AND prod_price >= 10;
```

1. 圆括号：求值顺序更高；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。
2. IN操作符：IN操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN取一组由逗号分隔、括在圆括号中的合法值。

```
WHERE vend_id IN ('DLL01','BRS01');
```

1. IN：WHERE子句中用来指定要匹配值的清单的关键字，功能与OR相当，语法更清楚直观，求值顺序更容易管理，执行更快，能更动态建立WHERE子句
2. NOT操作符：否定其后跟的任何条件。用在要匹配的列前

```
WHERE NOT vend_id = 'DLL01'
```

小结：

- 如何使用AND和OR操作符组合成WHERE子句
- 如何明确管理求值顺序
- 如何使用IN和NOT操作符

## 第6课 用通配符进行过滤

- 什么是通配符
- 如何使用通配符
- 怎样使用LIKE操作符进行通配搜索

### LIKE操作符

1. 通配符（wildcard）：用来匹配值的一部分的操作符
2. 搜索模式（search pattern）：由字面值、通配符或两者组合构成的搜索条件
3. 利用通配符，可以创建比较特定数据的搜索模式；通配符本身实际上是SQL的WHERE子句中有特殊含义的字符。在搜索子句中使用通配符，必须使用LIKE操作符
4. 谓词（predicate），从技术上说LIKE是谓语，而不是操作符

5. 通配符搜索只能用于文本字段（字符串），非文本数据类型字段不能使用通配符搜索

**百分号（%）通配符**

1. 在搜索中，%表示任何字符出现任意次数

```
WHERE prod_name LIKE ‘Fish%’;
```

此例子使用搜索模式’Fish%’，在执行这条子句时，将检索任意以Fish起头的值

1. 区分大小写：根据DBMS的不同及其配置，搜索可以是区分大小写的
2. 通配符可以在搜索模式的任意位置使用，并且可以使用多个通配符

```
WHERE prod_name LIKE ‘%bean bag%’;
```

1. %代表搜索模式中给定位置的0个、1个或多个字符
2. 注意NULL：

```
WHERE prod_name LIKE ‘%’
```

不会匹配产品名称为NULL的行

1. 下划线（_）通配符：匹配单个字符

```
WHERE prod_name LIKE ‘_ inch teddy bear’;
```

1. 使用%通配符：

```
WHERE prod_name LIKE ‘% inch teddy bear’;
```

能匹配多个字符，而_只能匹配一个字符

**方括号（[ ]）通配符**

1. 方括号（[ ]）通配符用来指定一个字符集，它必须匹配指定位置的一个字符。例如，找出所有名字以J或M起头的联系人：

```
WHERE cust_contact LIKE '[JM]%';
```

匹配方括号中任意一个字符，只能匹配单个字符；%匹配第一个字符之后的任意数目的字符

1. 脱字符：使用前缀字符脱字符来否定 。例如，'[^JM]%'不以字母J或M起头的联系人

**使用通配符的技巧**

1. 通配符搜索比其他搜索要耗费更长的处理时间
2. 不要过度使用通配符
3. 不要把通配符置于开始处，搜索最慢
4. 仔细注意通配符的位置

**小结：**

- 什么是通配符
- 如何在WHERE子句中使用SQL通配符

## 第7课 创建计算字段

- 什么是计算字段
- 如何创建计算字段
- 如何从应用程序中使用别名引用计算字段

**计算字段**

1. 存储在数据表中的数据一般不是应用程序所需要的格式，需要直接从数据库中检索出转换、计算或格式化过的数据
2. 计算字段是运行时在SELECT语句内创建的
3. 字段（ Field）：基本上与列（column）意思相同，术语字段通常与计算字段一起使用
4. 注意：只有数据库知道SELECT语句中哪些列是实际的表列，哪些列是计算字段

5. 客户端与服务器的格式：在服务器上完成数据的格式转换比在客户端上完成这些操作要快得多

### 拼接字段

1. 拼接（concatenate）：将值连接到一起构成单个值

```
SELECT vend_name + vend_country / SELECT Concat(vend_name, vend_country)
```

1. TRIM( )/RTRIM( )/LTRIM( )函数：去空格
2. 使用别名：别名（alias）是一个字段或值的替换名，用AS关键字赋予 别名有时候也称为导出列（derived column）

```
SELECT RTRIM(vend_name) + RTRIM(vend_country) AS vend_title
```

1. AS vend_title指示SQL语句创建一个包含指定计算结果的名为vend_title的计算字段，任何客户端都可以按名称引用这个列

### 执行算术计算

1. 计算字段：对检索出的数据进行算术计算

```
SELECT prod_id, quantity, item_price, quantity*item_price AS
expanded_price
```

1. SQL算术操作符：加（+）减（-）乘（*）除（/）
2. 如何测试计算：SELECT通常用于检索数据，但省略FROM子句后就是简单地访问和处理表达式
3. 可以根据需要使用SELECT语句进行检测：

```
SELECT Now( );
SELECT Trim( 'abc' );
SELECT 2*3;
```

### 小结：

- 什么是计算字段
- 如何创建计算字段
- 计算字段在字符拼接和算数计算中的用途
- 如何创建和使用别名，以便应用程序能引用计算字段

## 第8课 使用函数处理数据

- 什么是函数
- DBMS支持何种函数
- 如何使用这些函数
- 为什么SQL函数的使用会带来问题

### 函数

1. 函数一般在数据上执行的，为数据的转换和处理提供方便
2. 每一个DBMS都有特定的函数；虽然所有类型的函数一般都可以在每个DBMS中使用，但各个函数的名称和语法可能极其不同
3. DBMS函数的差异：例如，提取字符串的组成部分；数据类型的转换；取当前日期等函数在不同DBMS函数不同。
4. 可移植（portable）：所编写的代码可以在多个系统上运行
5. 与SQL语句不一样，SQL函数是不可移植的

### 使用函数

1. 大多数SQL实现支持以下类型的函数：

- 用于处理文本字符串（如删除或填充值，转换值为大写或小写）的文本函数；
- 用于数值数据上进行算术操作（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（如返回两个日期之差，检查日期有效性）的日期和时间函数；
- 返回正使用的特殊信息（如返回用户登录信息）的系统函数

1. 文本处理函数：

- 返回字符串的长度（LENGTH( )）；
- 将字符串转换为大写字母（UPPER( )）；
- 将字符串转换为小写字母（LOWER( )）；
- 返回字符串左边的函数（LEFT( )）；

- 去掉字符串左边的空格（LTRIM( )）；
- 返回字符串的SOUNDEX值（SOUNDEX是一个将任何文本串转换为描述其语音表示的字母数字模式的算法）

1. 日期和时间处理函数：日期和时间采用相应的数据类型存储在表中，每种DBMS都有自己的特殊形式
2. 应用程序一般不使用日期和时间的存储格式，因此日期和时间函数总是用来读取、统计和处理这些值。
3. 数值处理函数：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、三角或几何运算。在主要的DBMS的函数中，数值函数是最一致、最统一的函数
4. 常用的数值处理函数：

- 返回一个数的绝对值（ABS( )）
- 返回一个角度的余弦（COS( )）
- 返回一个角度的正弦（SIN( )）
- 返回一个角度的正切（TAN( )）
- 返回一个数的平方根（SQRT( )）
- 返回一个数的指数值（EXP( )）

**小结：**

- 如何使用SQL的数据处理函数
- 函数在各种DBMS中实现很不一致

## 第9课 汇总数据

- 什么是SQL的聚集函数
- 如何使用它们汇总表的数据

**聚集函数**

1. SQL查询可用于检索数据，利用聚集函数汇总数据，以便分析和报表生成
2. 需要汇总数据，而不需要实际数据本身的例子：找出表列（或所有行或某些特定的行）的最大值、最小值、平均值
3. 聚集函数（aggregate function）：对某些行运行的函数，计算并返回一个值
4. SQL聚集函数：

- 返回某列的平均值（AVG( )）
- 返回某列的行数（COUNT( )）
- 返回某列的最大值（MAX( )）
- 返回某列的最小值（MIN( )）
- 返回某列值之和（SUM( )）

1. AVG( )函数：通过对表中行数计数并计算其列值之和，求得该列的平均值。AVG( )函数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
```

1. 只能用于单个列：AVG( )只能用来确定特定数值列的平均值，而且列名必须作为函数参数给出
2. NULL值：AVG( )函数忽略值为NULL的行
3. COUNT( )函数：确定表中行的数目或符合特定条件的行的数目
4. 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
5. 只对具有电子邮件地址的客户计数：

```
SELECT COUNT(cust_email) AS num_cust
```

1. MAX( )函数返回指定列中的最大值，要求指定列名，忽略NULL值
2. MIN( )函数返回指定列的最小值，要求指定列名，忽略NULL值
3. SUM( )函数用来返回指定列值的和，也可以用来合计计算值

```
SELECT SUM(item_price*quantity) AS total_price
```

1. 聚集不同值：对所有行执行计算，指定ALL参数或不指定参数（ALL是默认行为）；只包含不同的值，指定DISTINCT参数

```
SELECT AVG(DISTINCT prod_price) AS avg_price
```

1. 组合聚集函数：SELECT语句可根据需要包含多个聚集函数

2. 取别名：在指定别名包含某个聚集函数的结果时，不应该使用表中的实际列名

小结：

- 聚集函数用来汇总数据
- SQL支持5个聚集函数，可以用多种方法使用它们，返回所需结果

## 第10课 分组数据

- 如何分组数据，以便汇总表内容的子集
- GROUP BY子句
- HAVING子句

聚集函数

1. 使用分组可以将数据分为多个逻辑组，对每个组进行聚集计算
2. 创建分组：GROUP BY子句指示DBMS分组数据，对每个组而不是整个结果集进行聚集
3. 统计每个供应商提供的产品数量：按供应商的ID对数据进行排序分组，利用COUNT(*)进行统计。会对每个vend_id而不是整个表进行计算num_prods

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

1. 规定：

- GROUP BY子句必须出现在WHERE子句之后，ORDER BY 子句之前；
- 如果分组列中包含有NULL值的行，将分为一组；
- 除聚集计算语句外，SELECT语句中的每一列都必须在GROUP BY子句中给出；
- 大多数SQL实现不允许GROUP BY列带有长度可变的数据类型（如文本或备注型字段）；
- GROUP BY子句中列出的每一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，则必须在子句中指定相同的表达式，不能使用别名；
- 如果在GROUP BY子句中嵌套了分组，数据将在最后指定的分组上进行汇总，在建立分组时，指定的列一起计算，不能从个别的列取回数据；

- GROUP BY子句可以包含任意数目的列，因而可以对分组进行嵌套，更细致地进行数据分组。

**过滤分组**

1. 除了GROUP BY分组数据之外，SQL还支持过滤分组，规定包括哪些分组，排除哪些分组
2. HAVING子句过滤分组：HAVING过滤分组，WHERE过滤指定的行而不是分组
3. HAVING和WHERE的差别： WHERE在数据分组前进行过滤，HAVING在数据分组后进行过滤
4. 列出具有两个以上产品且其价格大于等于4的供应商

```
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
WHERE prod_price >= 4
GROUP BY vend_id
HAVING COUNT(*) > 2;
```

1. 使用HAVING和WHERE：使用HAVING时应该结合GROUP BY子句，而WHERE子句用于标准的行级过滤

**分组和排序**

1. ORDER BY和GROUP BY的差别：

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

1. 一般在使用GROUP BY子句时，应该也给出ORDER BY子句，不要依赖GROUP BY排序数据
2. 检索包含三个或更多物品的订单号和订购物品的数目：使用GROUP BY子句按订单号分组数据，以便COUNT(*)函数能够返回每个订单中的物品数量；HAVING子句过滤数据，使得只返回包含三个或更多物品的订单；最后用ORDER BY子句排序输出

```
SELECT order_num, COUNT(*) AS items
```

```
FROM OrderItems
GROUP BY order_num
HAVING COUNT(*) >= 3;
ORDER BY items, order_num;
```

### SELECT子句顺序

1. SELECT：要返回的列或表达式，必须使用
2. FROM：从中检索数据的表，仅在从表中选择数据时使用
3. WHERE：行级过滤，不是必须使用
4. GROUP BY：分组说明，仅在按组计算聚集时使用
5. HAVING：组级过滤，不是必须使用
6. ORDER BY：输出排序顺序，不是必须使用

### 小结：

- 如何使用GROUP BY子句对多组数据进行汇总计算，返回每个组的结果
- 如何使用HAVING子句过滤特定的组
- ORDER BY与GROUP BY之间的差异
- WHERE与HAVING之间的差异

## 第11课 使用子查询

- 什么是子查询
- 如何使用子查询

### 子查询

1. 查询（query）:任何SQL语句都是查询，但此术语一般指SELECT语句，从单个数据表中检索数据的单条语句
2. 子查询（subquery）：嵌套在其他查询中的查询

### 利用子查询进行过滤

1. 列出订购物品RGAN01的所有顾客：检索包含物品RGAN01的所有订单的编号

```
SELECT order_num FROM OrderItems WHERE prod_id = 'RGAN01';
```

检索具有前一步骤列出的所有的订单编号的所有顾客的ID

```
SELECT cust_id FROM Customers WHERE order_num IN (20007,20008);
```

检索前一步骤返回的所有顾客ID的顾客信息

```
SELECT cust_name, cust_contact FROM Customers WHERE cust_id IN
('1000004','1000005');
```

1. 在WHERE子句中使用子查询能够编写出功能很强且很灵活的SQL语句。对于能嵌套的子查询的数目没有限制，不过在实际使用中由于性能的限制，不能嵌套太多的子查询。
2. 作为子查询的SELECT语句只能查询单个列，企图检索多个列将返回错误

**作为计算字段使用子查询**

1. 显示Customers表中每个顾客的订单总数。执行这个操作，需要遵循下面的步骤：

- 从Customers表中检索顾客列表；
- 对于检索出的每个顾客，统计其在Orders表中的订单数目

```
SELECT cust_name, cust_state,
       (SELECT COUNT(*)
       FROM Orders
       WHERE Customers.cust_id = Orders.cust_id) AS order_nums
FROM Customers
ORDER BY cust_name;
```

1. 完全限定列名：指定表名和列名，语法上用一个句点分隔表名和列名
2. 如果在SELECT语句中操作多个表，就应使用完全限定列名来避免歧义，没有具体指定会返回错误结果

**小结：**

- 什么是子查询
- 如何使用子查询

- 子查询用于WHERE子句中的IN操作符
- 子查询用来填充计算列

## 第12课 联结表

- 什么是联结
- 为什么使用联结
- 如何编写使用联结的SELECT语句

### 联结

1. SQL最强大的功能之一就是能在数据查询的执行中联结（join）表，联结是利用SQL的SELECT能执行的最重要的操作

### 关系表

1. 关系表的设计就是要把信息分解成多个表，一类数据一个表，各表通过某些共同的值互相关联（所以才叫关系数据库）
2. 可伸缩（scale）：能够适应不断增加的工作量而不失败，设计良好的数据库或应用程序称为可伸缩性好

### 为什么使用联结

1. 将数据分解为多个表能更有效地存储，更方便地处理，并且可伸缩性更好
2. 将数据存储在多个表中，通过联结检索出数据。简单说，联结是一种机制，用来在一条SELECT语句中关联表，因此称为联结
3. 使用特殊的语法，可以联结多个表返回一组输出，联结在运行时关联表中正确的行

### 创建联结

1. 创建联结：指定要联结的所有表以及要关联它们的方式

```
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products
WHERE Vendors.vend_id = Products.vend_id;
```

1. SELECT语句指定要关联的表的名字，WHERE子句指示DBMS将Vendors表中的vend_id与Products表中的vend_id匹配起来

2. WHERE子句的重要性：
3. 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没有指示DBMS如何对表进行联结。
4. 在联结两个表时，实际要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，只包含那些匹配给定条件（联结条件）的行
5. 笛卡尔积（cartesian product）：由没有联结条件的表关系返回的结果为笛卡尔积，也称为叉联结（cross join）。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
6. 要保证所有联结都有WHERE子句，同时保证WHERE子句的正确性

### 内联结

1. 等值联结 / 内联结（inner join）：基于两个表之间的相等测试，可以对这种联结使用不同的语法，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;
```

1. 联结条件用特定的ON子句而不是WHERE子句给出，传递给ON的实际条件与传递给WHERE的相同

### 联结多个表

1. SQL不限制一条SELECT语句中可以联结的表的数目，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
2. 显示订单20007中的物品：

```
SELECT prod_name, vend_name, prod_price, quantity
FROM Vendors, OrderItems, Products
WHERE Vendors.vend_id = Products.vend_id
 AND OrderItems.prod_id = Products.prod_id
 AND order_num = 20007;
```

1. DBMS在运行时关联指定的每个表，以处理联结，这种处理非常耗费资源。联结的表越多，性能下降越厉害
2. 执行任一给定的SQL操作一般不止一种方法，很少有绝对正确或绝对错误的方法。性能可能会受操作类型、所使用的DBMS、表中的数据量、是否存在索引

或键等条件的影响

小结：

- 联结是SQL中一个最重要、最强大的特性，有效地使用联结需要对关系数据库设计有基本的了解
- 关系数据库设计的基本知识
- 什么是等值联结

## 第13课 创建高级联结

- 如何使用表别名
- 如何对被联结的表使用聚集函数

### 使用表别名

1. SQL除了可以对列名和计算字段使用别名，还允许给表名起别名：可以缩短SQL语句；允许在一条SELECT语句中多次使用相同的表

```
SELECT cust_name, cust_contact
FROM Customers AS C, Orders AS O, OrderItems AS OI
WHERE C.cust_id = O.cust_id
    AND O.oder_num = OI.order_num
    AND prod_id = 'RGAN01';
```

1. 表别名只在查询执行中使用，与列表名不一样，表别名不返回到客户端

### 使用不同类型的联结

1. 自联结（self-join）：使用表别名的一个重要的原因是能在一条SELECT语句中不止一次引用相同的表；自联结通常作为外部语句，用来替代从相同表中检索数据的子查询语句
2. 要给与Jim Jones同一公司的所有顾客发送一封信件：找出Jim Jones工作的公司，然后找出该公司工作的顾客。

```
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';
```

2. SELECT语句使用明确给出所需列的全名，指示DBMS哪一列（其实它们是同一列）；WHERE子句联结两个表，然后按第二个表中cust_contact的过滤数据，返回所需的数据
3. 自然联结（natural join）：标准的联结返回所有数据，相同的列甚至多次出现；自然联结排除多次出现，使每一列只返回一次
4. 自然联结要求你只能选择那些唯一的列，一般通过对一个表使用通配符（SELECT *），而对其他表的列使用明确的子集来完成
5. 很少有不是自然联结的内联结，基本上每个内联结都是自然联结
6. 外联结（outer join）：许多联结将一个表中的行与另一个表中的行相关联，但有时候需要包含没有关联行的那些行，这种联结称为外联结
7. 使用内联结，检索所有顾客及其订单；使用外联结，检索包括没有顾客在内的所有顾客
8. 选择FROM子句左边的表（Customers表）中选择所有行
9. 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）
10. 外联结的类型：左外联结和右外联结之间唯一的差别是所关联的表的顺序。调整FROM或WHERE子句中表的顺序，左外联结可以转换成右外联结
11. 全外联结（full outer join）：检索两个表中的所有行并关联那些可以关联的行，全外联结包含两个表的不关联的行

**使用带聚集函数的联结**

1. 例子：要检索所有顾客及每个顾客所下的订单数

```
SELECT Customers.cust_id, COUNT(Orders.order_num) AS num_orders
FROM   Customers INNER JOIN Orders
ON  Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

1. 这条SELECT语句使用INNER JOIN将Customers和Orders表互相关联；GROUP BY子句按顾客分组数据，因此函数调用COUNT(Orders.order_num)对每个顾客的订单计数，将它作为num_ord返回

**使用联结和外联结条件**

1. 注意所使用的联结类型

2. 关于确切的联结语法，应该查看具体文档，看相应的DBMS支持何种语法
3. 保证使用正确的联结条件，否则会返回不正确的数据
4. 总是提供联结条件，否则会得出笛卡尔积
5. 在一个联结中可以包含多个表，甚至可以对每个联结采用不同的联结类型

**小结：**

- 如何以及为什么使用别名
- 不同的联结类型以及每类联结所使用的语法
- 如何与联结一起使用聚集函数
- 使用联结时应该注意的问题

## 第14课 组合查询

如何利用UNION操作符将多条SELECT语句结合组合成一个结果集

**组合查询**

1. 组合查询（union）或复合查询（compound query）：多数SQL语句只包含一个或多个从表中返回数据的单条SELECT语句；SQL也允许执行多个查询，并将结果作为一个查询结果集返回
2. 什么时候使用组合查询：

- 在一个查询中从不同的表返回数据结构；
- 对一个表执行多个查询，按一个查询返回数据

1. 组合条件和多个WHERE查询：任何具有多个WHERE子句的SELECT语句都可以作为一个组合查询

```
SELECT cust_name, cust_contact, cust_email
FROM   Customers
WHERE  cust_state IN ('IL','IN','MI')
  OR  cust_name = 'Fun4A11';
```

1. 创建组合查询例子

```
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 = 'Fun4A11';
```

1. 使用UNION： 使用UNION很简单，给出每条SELECT语句，在各条语句之间放上关键词UNION
2. UNION指示DBMS执行这两条SELECT语句，并把输出组合成一个查询结果集。
3. 使用UNION组合SELECT语句的数目，SQL没有标准限值，最好参考一下具体的DBMS文档

### UNION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
- UNION中的每个查询必须包含相同的列、表达式或聚集函数（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可以隐含转换的类型

### 包含或取消重复的行

1. UNION从查询结果集中自动去除重复的行
2. UNION ALL：使用UNION ALL返回所有的匹配行，DBMS不取消重复的行

```
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 = 'Fun4A11';
```

### 对组合查询结果排序

1. 使用UNION组合查询时，只能使用一条ORDER BY子句，它必须位于最后一条SELECT语句之后；对于结果集，不存在用一种方式排序一部分，而用另一种方式排序另一种部分的情况

```
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 = 'Fun4A11'
ORDER BY cust_name, cust_contact;
```

小结：

- 如何用UNION操作符来组合SELECT语句

## 第15课 插入数据

如何利用SQL的INSERT语句将数据插入表中

### 组合查询

1. INSERT语句：用来将行插入到数据库表
2. 插入方式：

- 插入完整的行；
- 插入行的一部分；
- 插入某些查询的结果

1. 系统安全：使用INSERT语句可能需要客户端 / 服务器DBMS中的特定安全权限
2. 插入完整的行：使用基本的INSERT语法，要求指定表名和插入到新行中的值
   INSERT INTO 表名 VALUES 各列值
3. 存储到表中每一列的数据在VALUES子句中给出，必须给每一列提供一个值。如果某列没有值，则应该使用NULL值；各列必须以它们在表定义中出现的次序填充
4. INTO关键字：是可选的，为保证SQL代码在DBMS之间可移植，最好提供这个关键字
5. 安全性：应该避免使用这种语法，高度依赖表中列的定义次序，不能保证各列在下一次表结构变动后保持完全相同的次序
6. 推荐方法：在表名后的括号里明确给出列名。在插入行时，DBMS用VALUES列表中的相应值填入到列表中的对应项。VALUES必须以其指定的次序提供其指定的次序匹配指定的列名，不一定按各列出现在表中的实际次序。优点是，表的

结构改变，这条INSERT语句仍然能正确工作

```
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country,
                      cust_contact,
                      cust_email)
VALUES('1000000006',
       'Toy Land',
       '123 Any Street',
       'New York',
       'NY',
       '11111',
       'USA',
       NULL,
       NULL);
```

1. 不要使用没有明确给出列的INSERT语句
2. 不管使用哪种INSERT语法，VALUES的数目都必须正确

### 插入部分行

1. 省略列：如果表的定义允许，INSERT语法可以省略列，这表示可以给某些列提供值，给其他列不提供值

```
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country)
VALUES('1000000006',
       'Toy Land',
       '123 Any Street',
       'New York',
       'NY',
       '11111',
       'USA');
```

1. 省略列的条件：该列定义为允许NULL值；在表定义中给出默认值
2. 如果表中不允许有NULL值或默认值，这时省略了表中的值，DBMS会产生错误消息，相应的行不能成功插入

**插入检索出的数据**

1. INSERT SELECT：由一条INSERT语句和一条SELECT语句组成
2. 例子：把另一表中的顾客列合并到Customers表中

```
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country,
                      cust_contact,
                      cust_email)
SELECT  cust_id,
        cust_name,
        cust_address,
        cust_city,
        cust_state,
        cust_zip,
        cust_country,
        cust_contact,
        cust_email
FROM CustNew;
```

1. 从一个名为CustNew的表中读出数据并插入到Customers表。为了验证这个例子，首先需要创建和填充CustNew表，两个表的结构需要相同，填充数据时，不应该使用已经在Customers中用过的cust_id值。SELECT语句从CustNew检索出要插入的值，而不是列出它们；SELECT中列出的每一列对应于Customers表名后所跟的每一列。如果CustNew表中没有数据，则没有行被插入，但是操作仍然是合法的；如果表中有数据，则所有数据将被插入到Customers中
2. INSERT SELECT中的列名：DBMS不关心SELECT返回的列名，它使用的是列的位置
3. INSERT SELECT中的SELECT语句可以包含WHERE子句，以过滤插入的数据

**从一个表复制到另一个表**

1. SELECT INTO语句：将一个表的内容复制到另一个全新的表（运行中创建的表）
2. 与INSERT SELECT将数据添加到一个已经存在的表不同，SELECT INTO将数据复制到一个新表
3. INSERT SELECT与SELECT INTO的重要差别：前者导出数据，后者导入数据

```
SELECT *
INTO CustCopy
FROM Customers;
```

1. SELECT语句创建一个名为CustCopy的新表，并把Customers表的整个内容复制到新表中。因为这里使用SELECT*，所以将在CustCopy表中创建（并填充）与Customers表的每一列相同的列。要想只复制部分的列，可以明确给出列名，而不是使用*通配符
2. 使用SELECT INTO的注意事项：

- 任何SELECT选项和子句都可以使用，包括WHERE和GROUP BY；
- 可利用联结从多个表插入数据；
- 不管从多少个表中检索数据，数据都只能插入到一个表中

1. 进行表的复制：在复制的数据上测试SQL代码，不会影响实际数据

**小结：**

- 如何将行插入到数据库表中
- INSERT的几种方法，为什么要明确使用列名
- 如何利用INSERT SELECT从其他表中导入行
- 如何用SELECT INTO将行导出到一个新表

## 第16课 更新和删除数据

如何利用UPDATE和DELETE语句进一步操作表数据

**更新数据**

1. 使用UPDATE的方式：

- 更新表中的特定行；

- 更新表中的所有行

1. 安全权限：在客户端 / 服务器的DBMS中，使用UPDATE语句需要特殊的安全权限
2. 基本的UPDATE语句：

- 要更新的表(UPDATE)；
- 列名和新值(SET)；
- 确定要更新哪些行的过滤条件（WHERE）

```
UPDATE Customers
SET cust_email = 'kim@thetoystore.com'
WHERE cust_id = '1000000005';
```

1. UPDATE语句指定要更新的表名；SET命令将新值赋予给被更新的列；WHERE子句告诉DBMS更新哪一行，没有WHERE子句，DBMS会更新表中的所有行
2. 更新多个列：用一条SET命令，每个列值对之间用逗号分隔

```
UPDATE Customers
SET cust_email = 'sam@toyland.com',
    cust_contact = 'Sam Roberts'
WHERE cust_id = '1000000005';
```

1. UPDATE语句中可以使用子查询，使得能用SELECT语句检索出的数据更新列数据
2. 要删除某个列的值，可设置它为NULL，NULL表示没有值

```
UPDATE Customers
SET cust_email = NULL
WHERE cust_id = '1000000005';
```

**删除数据**

1. 使用DELETE的方式：从表中删除特定的行；从表中删除所有行
2. 不要省略WHERE子句：会不小心删除表中所有行
3. 安全性：在客户端 / 服务器的DBMS中，使用DELETE语句可能需要特殊的安全

权限

4. DELETE FROM要求指定从中删除数据的表名，WHERE子句过滤要删除的行，如果省略WHERE子句，将删除表中的每个顾客

```
DELETE FROM Customers
WHERE cust_id = '100000006';
```

1. DELETE不需要列名或通配符，DELETE删除整行而不是删除列，要删除指定的列，请使用UPDATE语句
2. 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

**更新和删除的指导原则**

1. 使用UPDATE或DELETE时所遵循的重要原则：

- 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；
- 在UPDATE或DELETE语句使用WHERE子句前，应该先使用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，这样DBMS不允许删除其数据与其他表相关联的行；
- 有的DBMS允许数据库管理员施加约束，防止执行不带WHERE子句的UPDATE或DELETE语句

**小结：**

- 如何使用UPDATE和DELETE语句处理表中的数据、
- 保证数据安全而应该遵循的一些指导原则

## 第17课 创建和操纵表

创建、删除和更改表的基本知识

**创建表**

1. SQL不仅用于表数据操纵，还用来执行数据库和表的所有数据，包括表本身的

创建和处理

2. 创建表的方法：

- 多数DBMS都具有交互式创建和管理数据库表的工具；表也可以直接用SQL语句操纵
- 使用界面工具会自动生成并执行相应的SQL语句

**表创建基础**

1. CREATE TABLE语句：

- 新表的名字，在CREATE TABLE关键字之后给出；
- 表列的名字和定义，用逗号分隔；
- 有的DBMS还要求指定表的位置

```
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 VARCHAR(1000) NULL
);
```

1. 表名紧跟CREATE TABLE关键字，实际的表定义（所有列）括在圆括号之中，各列之间用逗号分隔；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；整条语句以圆括号后的分号结束

**语句格式化：**

1. SQL语句格式化，代码安排在多个行上，列定义进行恰当的缩进，更易阅读和编辑
2. 替换现有的表：在创建新的表时，指定的表名必须不存在，否则会出错

**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值的列不接受没有列值的行，在插入或更新行时，该列必须有值
2. 每个表列要么是NULL值，要么是NOT NULL列，这种状态在创建时由表的定义规定

```
CREATE TABLE Vendors
(
  order_num INTEGER NOT NULL,
  order_date DATETIME NOT NULL,
  cust_id CHAR(10) NOT NULL
);
```

1. 如果插入没有值的列，将返回错误，且插入失败
2. NULL为默认设置，如果不指定NOT NULL，就认为指定的是NULL

```
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) ,
);
```

1. 主键和NULL值： 允许NULL值的列不能作为唯一标识
2. 指定默认值：SQL允许指定默认值，在插入行时如果不给出值，DBMS将采用默认值

```
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
);
```

1. 默认值经常用于日期和时间戳列：

- Access(NOW( ))
- DB2(CURRENT_DATE)
- MySQL(CURRENT_DATE( ))

- Oracle(SYSDATE)
- PostgreSQL(CURRENT_DATE)
- SQL Server(GETDATE( ))
- SQLite(date('now'))

**更新表**

1. ALTER TABLE语句：更新表定义，可以使用ALTER TABLE语句
2. 使用ALTER TABLE时需要考虑：

- 不要在表中包含数据时对其进行更新；
- 所有的DBMS都允许给现有的表增加列，不过对所增加列的数据类型有所限制；
- 许多DBMS不允许删除或更改表中的列；
- 多数DBMS允许重新命名表中的列；
- 许多DBMS限制对已经填有数据的列进行更改，对未填有数据的列几乎没有限制

1. 使用ALTER TABLE更改表结构时

- 必须给出要更改的表名；
- 列出要做哪些更改

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

```
ALTER TABLE Vendors
DROP COLUMN vend_phone;
```

1. 复杂表的删除步骤：

- 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；
- 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

1. 删除表

```
DROP TABLE CustCopy;
```

1. 删除表没有确认，也不能撤销，执行这条语句将永久删除该表
2. 使用关系规则防止意外删除：如果对某个表发布一条DROP TABLE语句，且该表是某个关系的组成部分，则DBMS将阻止这条语句的执行，直到该关系被删除为止

**重命名表**

1. 所有重命名操作的基本语法都要求指定旧表名和新表名

**小结：**

- CREATE TABLE用来创建新表
- ALTER TABLE用来更改表列（或其他诸如约束或索引等对象）
- DROP TABLE用来完整删除一个表

## 第18课 使用视图

- 什么是视图
- 视图是怎么工作的
- 何时使用视图
- 如何利用视图简化某些SQL操作

**视图**

1. 视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询
2. 例子：检索订购某种产品的顾客的查询，需要理解表的结构，知道如何创建查询和对表进行联结

```
SELECT cust_name, cust_contact
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Order.cust_id
```

```
  AND OrderItems.order_num = Orders.order_num
  AND prod_id = 'RGAN01';
```

1. 视图则是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 = 'RGAN01';
```

视图的作用：轻松地检索出相同的数据。ProductCustomers是一个视图，它不包含任何列或数据，包含的是一个查询

**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其基本查询细节
- 使用表的一部分而不是整个表
- 保护数据，可以授予用户访问表的特定部分权限，而不是整个表的访问权限
- 更改数据格式的表示。视图可返回与底层表的表示和格式不同的数据
- 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，甚至添加和更新数据

1. 视图本身不包含数据，因此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

**视图的规则和限制**

1. 创建视图时的限制，需要查看具体的DBMS文档
2. 常见的规则和限制：

- 与表一样，视图必须唯一命名
- 对于可以创建的视图数目没有限制
- 创建视图，必须具有足够的访问权限
- 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视图
- 许多DBMS禁止在视图查询中使用ORDER BY子句

- 有些DBMS要求返回所有例进行命名，如果列是计算字段，则需要使用别名
- 视图不能索引，也不能有关联的触发器或默认值
- 有些DBMS把视图作为只读的查询，这表示可以从视图检索数据，但不能将数据写会底层表
- 有些DBMS允许创建这样的视图，它不能进行导致行不再属于视图的插入或更新。例如有一个视图，只检索带有电子邮件地址的顾客。如果更新某个顾客，删除他的电子邮件地址，将使该顾客不再属于视图。这是默认行为，而且是允许的，但有的DBMS可能会防止这种情况发生

**创建视图**

1. 创建视图时的限制，需要查看具体的DBMS文档
2. CREATE VIEW语句：只能用于创建不存在的视图。覆盖（或）更新视图，必须先删除它，然后再重新创建。删除视图，可以使用DROP语句，语法为DROP VIEW viewname

```
CREATE VIEW ProductCustomers AS
SELECT cust_name, cust_contact
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Order.cust_id
 AND OrderItems.order_num = Orders.order_num
```

```
SELECT cust_name, cust_contact
FROM ProductCustomers
WHERE prod_id = 'RGAN01';
```

1. 利用视图简化复杂的联结：创建一个名为ProductCustomers的视图，它联结三个表，返回已订购了任意产品的所有顾客的列表。如果执行SELECT *FROM ProductCustomers ，将列出订购了任意产品的顾客
2. 这条语句通过WHERE子句从视图中检索特定数据。当DBMS处理此查询时，它将指定的WHERE子句添加到视图查询中已有的WHERE子句中，以便正确过滤数据
3. 视图极大地简化了复杂SQL语句的使用。利用视图，可一次性编写基础的SQL，然后根据需要多次使用
4. 用视图重新格式化检索出的数据：假设经常需要某种格式的结果，我们不必在每次需要时执行这种拼接，而是创建一个视图，使用视图即可
5. 要检索数据，创建所有的邮件标签：

```
CREATE VIEW VendorLocation AS
SELECT RTRIM(vend_name) + '(' + RTRIM(vend_country) + ')'
FROM Vendors;
```

```
SELECT * FROM VendorLocations;
```

1. SELECT约束全部适用
2. 用视图过滤不想要的数据：可以定义CustomerEmailList视图，过滤没有电子邮件地址的顾客

```
CREATE VIEW CustomerEmailList AS
SELECT cust_id, cust_name, cust_email
FROM Customers
WHERE cust_email IS NOT NULL;
```

```
SELECT *
FROM CustomerEmailList;
```

1. 使用视图和计算字段：先转换为视图，然后再检索订单的详细内容

```
CREATE VIEW OrderItemExpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM OrderItems;
```

```
SELECT *
FROM OrderItemExpanded
WHERE order_num = 20008;
```

小结：

- 视图为虚拟的表，包含的不是数据而是根据需要检索数据的查询

- 视图提供了一种封装SELECT语句的层次，可用来简化数据处理，重新格式化或保护基础数据

## 第19课 使用存储过程

- 什么是存储过程
- 为什么要使用存储过程
- 如何使用存储过程
- 创建和使用存储过程的基本语法

### 存储过程

1. 有一些复杂的操作需要多条语句才能完成。例如:

- 为了处理订单，必须核对以保证库存中有相应的物品；
- 如果物品有库存，需要预定，不再出售给别人，并且减少物品数据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

1. 可以创建存储过程。存储过程就是为以后使用而保存的一条或多条SQL语句，可将其视为批处理文件，虽然他们的作用不仅限于批处理

### 为什么要使用存储过程

- 通过把处理封装在一个易用单元内，可以简化复杂的操作
- 由于不要求反复建立一系列处理步骤，因而保证了数据的一致性。如果所有开发人员和应用程序都使用同一存储过程，则所使用的代码都是相同的
- 简化对变动的管理。这一点延伸就是安全性。通过存储过程限制对基础数据的访问，减少数据讹误（无意识的或别的原因所导致的数据讹误）的机会。
- 因为存储过程通常以编译过的形式存储，所以DBMS处理命令所需的工作量少，提高了性能
- 存在一些只能用在单个请求中的SQL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

1. 缺陷:

- 不用的DBMS中存储过程语法有所不同。编写真正的可移植的存储过程几乎是不可能的。存储过程的自我调用（名字以及数据如何传递）可以相对保持可移植；

- 编写存储过程要比编写基本SQL语句复杂

**执行存储过程**

1. EXECUTE接受存储过程名和需要传递给它的任何参数。

```
EXECUTE AddNewProduct('JTS01',
                      'Stuffed Effiel Tower',
                      6.49,
                      'Plush stuffed toy with the text La Tour Eiffel
in red white and blue');
```

2. 这里执行一个名为AddNewProduct的存储过程，将一个新产品添加到Products表中。AddNewProduct有四个参数分别是：供应商ID（ Vendors表的主键）、产品名、价格和描述。这4个参数匹配存储过程中4个预期变量（定义为存储过程自身的组成部分）。此存储过程将新行添加到Products表中，并将传入的属性赋给相应的列。Products表中的主键不作为属性传递给存储过程，是想使生成此ID的过程自动化
3. 存储过程执行的基本形式：

- 验证传递的数据，保证所有4个参数都有值
- 生成用作主键的唯一ID
- 将新产品插入Products表，在合适的列中存储生成的主键和传递的数据

1. 不同DBMS的执行选择：

- 参数可选，具有不提供参数时的默认值
- 不按次序给出参数，以“参数=值”的方式给出参数
- 输出参数，允许存储过程在正执行的应用程序中更新所用的参数
- 用SELECT语句检索数据
- 返回代码，允许存储过程返回一个值到正在执行的应用程序

**创建存储过程**

1. 例子：对邮件发送清单中具有邮件地址的顾客进行计数。此存储过程没有参数。调用程序检索SQL Server的返回代码提供的值，用DECLARE语句声明了一个名为@cnt的局部变量（SQL Server中所有局部变量名都以@起头）；然后在SELECT 语句中使用这个变量，让它包含COUNT( )函数返回的值；最后用RETURN @cnt语句将计数返回给调用程序

2. 调用SQL Server：这段代码声明了一个变量用来保存存储过程返回的任何值，然后执行存储过程，再使用SELECT语句显示返回的值。
3. 3、具体例子见书本p173

小结：

- 什么是存储过程
- 为什么使用存储过程
- 执行和创建存储过程的语法
- 使用存储过程的一些方法

## 第20课 管理事务处理

- 什么是事务处理
- 如何利用COMMIT和ROLLBACK语句管理事务处理

事务处理

1. 事务处理（transaction processing）:使用事物处理，通过确保成批的SQL操作要么完全执行，要么完全不执行，来维护数据库的完整性
2. 关系数据库把数据存储在多个表中，使数据更容易操纵、维护和重用
3. 给系统添加订单的过程：

- 检查数据库中是否存在相应的顾客，如果不存在则添加；
- 检索顾客的ID；
- 在 Orders表添加一行，与顾客ID相关联；
- 检索Orders表中赋予的新订单ID；
- 为订购的每个物品在OrderItems表中添加一行，通过检索出来的ID把它与Orders表关联（并且通过产品ID与Products表关联）。

5. 若是数据库故障，这个过程无法完成，就需要使用事物处理
6. 事物处理：是一种机制，用来管理必须成批执行的SQL操作，保证数据库不包含不完整的操作结果
7. 利用事物处理，可以保证一组操作不会中途停止，它们要么完全执行，要么完全不执行
8. 说明这一过程是如何工作的：

- 检查数据库中是否存在相应的顾客，如果不存在，则添加他；

- 提交顾客信息
- 检索顾客的ID
- 在Orders表中添加一行
- 如果向Orders表添加行时出现故障，回退
- 检索Orders表中赋予的新订单ID
- 对于订购的每项物品，添加新行到OrderItem表
- 如果向OrderItems添加行时出现故障，回退所有添加的OrderItems行和Orders行

1. 事物处理的术语：

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

**控制事务处理**

1. 不同DBMS用来实现事务处理的语法不同，在使用事务处理时请参阅相应的DBMS文档
2. 管理事务的关键在于将SQL语句分解为逻辑块，并明确规定数据何时应该回退，何时不应该回退

```
BEGIN TRANSACTION
...
COMMIT TRANSACTION
```

在这个例子中两个语句之间的SQL必须完全执行或完全不执行

1. 使用ROLLBACK撤销SQL语句：

```
DELETE FROM Orders;
ROLLBACK;
```

在此例子中，执行DELETE操作，然后用ROLLBACK语句撤销。这个例子说明在事务处理块中，DELETE操作与UPDATE操作和INSERT操作一样，并不是最终结果

1. 使用 COMMIT操作：一般的SQL语句都是针对数据库表直接执行和编写的，这就是所谓的隐式提交（implicit commit），即提交（写或保存）操作是自动进行的
2. 在事务处理块中，提交不会隐式进行。
3. 在这个SQL Server中，从系统完全删除订单12345。因为涉及更新两个数据库表Orders和OrderItems，所以使用事务处理块来保证订单不被部分删除。最后的COMMIT语句仅在不出错时更改。如果第一条DELETE起作用，但第二条失败，则DELETE不会提交

```
BEGIN TRANSACTION
DELETE OrderItems WHERE order_num = 12345
DELETE Orders WHERE order_num = 12345
COMMIT TRANSACTION
```

1. 使用保留点：使用简单的ROLLBACK和COMMIT语句，就可以写入或撤销整个事务。但是，只对简单的事务才能这么做，复杂的事务需要部分提交或回退
2. 要支持回退部分事务，必须在事务处理块中的合适位置放置占位符。这样，如果需要回退，可以回退到某个占位符。在SQL中，这些占位符称为保留点
3. 在SQL Server中可以如下进行

```
SAVE TRANSACTION delete1;
```

1. 下图是一个完整的SQL Server的例子，在SQL 代码中设置任意多的保留点，越多越好，可以任意回退。

```
BEGIN TRANSACTION
INSERT INTO Customers(cust_id, cust_name) VALUES ('10000000010','Toy
Emporium');
SAVE TRANSACTION StartOrder;
INSERT INTO Orders(order_num, order_date, cust_id) VALUES (20100,
'2001/12/1','10000000010');
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num, order_item, prod_id, quantity,
item_price) VALUES (20100, 2, 'BR03', 100, 10.99);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
COMMIT TRANSACTION;
```

小结：

- 事务是必须完整执行的SQL语句块
- 如何使用COMMIT和ROLLBACK语句对何时写数据、何时撤销进行明确管理
- 如何使用保留点，更好地控制回退操作

## 第21课 使用游标

- 什么是游标
- 如何使用游标

### 游标

1. SQL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）
2. 结果集（result set）：SQL查询所检索出的结果
3. 游标（cursor）：是一个存储在DBMS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。在存储游标之后，应用程序可以根据需要滚动或浏览其中的数据
4. 游标的作用：在检索出来的行中前进或后退一行或多行
5. 游标常见的一些选项和特性：

- 能够标记游标为只读，使数据能读取，但不能更新和删除
- 能控制可以执行的定向操作（向前、向后、第一、最后、绝对位置、相对位置等）
- 能标记某些列为可编辑的，某些列为不可编辑的
- 规定范围，使游标对创建它的特定请求或对所有请求可访问
- 指示DBMS对检索出的数据进行复制，使数据在游标打开和访问期间不变化

1. 游标主要用于交互式应用，其用户需要滚动屏幕上的数据，并对数据进行进行浏览或做出更改。因为游标对基于Web的应用（ASP/ASP.NET/ColdFusion/PHP/Python/Ruby/JSP等）用处不大，因为应用服务器是数据库客户而不是最终用户

### 使用游标

1. 使用游标涉及几个明确的步骤：

- 在使用游标前，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语句和游标选项

- 一旦声明，就必须打开游标以供使用。这个过程用前面定义的SELECT语句把实际数据检索出来
- 对填有数据的游标，根据需要检索各行
- 在结束游标使用时，必须关闭游标，可能的话，释放游标

**创建游标：**

1. 使用DECLARE语句创建游标，DECLARE命名游标，并定义相应的SELECT语句，根据需要带WHERE子句和其他子句
2. 例子，创建一个游标来检索没有电子邮件地址的所有顾客，作为应用程序的组成部分，帮助操作人员找出空缺的电子邮件地址

```
DECLARE CustCursor CURSOR
FOR
SELECT * FROM Customers
WHERE cust_email IS NULL
```

**使用游标：**

1. 使用OPEN CURSOR语句打开游标

```
OPEN CURSOR CustCursor
```

1. 分析：在处理OPEN CURSOR语句时，执行查询，存储检索出的数据以供浏览和滚动
2. FETCH语句：使用FETCH语句访问游标数据，指出要检索哪些行，从何处检索它们以及将它们放于何处

```
DECLARE TYPE CustCursor IS REF CURSOR
    RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE
BEGIN
    OPEN CustCursor;
    FETCH CustCursor INTO CustRecord;
    CLOSE CustCursor;
END;
```

1. FETCH用来检索当前行，放到声明的变量CustRecord中。对于检索出来的数据不做任何处理
2. 关闭游标：游标在使用完毕时需要关闭 CLOSE CustCursor

小结：

- 什么是游标
- 为什么使用游标
- 使用的DBMS可能会提供某种形式的游标

## 第22课 高级SQL特性

高级数据处理特性：约束、索引、触发器

### 约束

1. 引用完整性（referential integrity）：关系数据库存储分解为多个表的数据，每个表存储相应的数据，利用键来建立一个表到另一个表的引用
2. 正确地进行关系数据库设计，需要一种方法保证只在表中插入合法数据
3. 最好不要在插入新行时进行检查：会有一些客户端不实施这些规则；执行客户端检查是非常耗时的，而DBMS执行这些检查会相对高效。
4. 约束（constraint）：管理如何插入或处理数据库数据的规则
5. DBMS通过在数据库表上施加约束来实施引用完整性，大多数约束是在表定义中定义的

### 主键

1. 主键：是一种特殊的约束，用来保证每一列中的值是唯一的，而且永不改动，表中的一列的值唯一标识表中的每一行

```
CREATE TABLE Vendors
(
    vend_id       CHAR(10) NOT NULL PRIMARY KEY,
    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
);
```

1. 使用CONSTRAINT语法，给表的vend_id列添加关键字PRIMARY KEY，使其成为主键

```
ALTER TABLE Vendors
ADD CONSTRAINT PRIMARY KEY (vend_id);
```

外键

1. 外键：外键是表中的一列，其值必须列在另一表的主键中，外键是保证引用完整的极其重要的部分
2. 例子：Orders中顾客ID不一定是唯一的；如果某个顾客有多个订单，则有多个行具有相同的顾客ID（每个订单都有不同的订单号）。在Orders的顾客ID列上定义了一个外键，该列只能接受Customers表的主键值

```
CREATE TABLE Orders
(
  order_num   INTEGER  NOT NULL PRIMARY KEY,
  order_date  DATETIME NOT NULL,
  cust_id     CHAR(10) NOT NULL REFERENCES Customers (cust_id)
);
```

1. 表定义使用了REFERENCES关键字，表示cust_id中的任何值都必须是Customers中的cust_id中的值
2. 使用CONSTRAINT语法来完成：

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

1. 外键有助防止意外删除：在定义外键后，DBMS不允许删除在另一个表中具有关联行的行。例如，不能删除关联订单的顾客。删除该顾客的唯一方法是首先删除相关的订单。由于需要一系列的删除，因而利用外键可以防止意外删除数据
2. 级联删除（cascading delete）：有的DBMS支持该特性，允许在一个表中删除行时删除所有相关特性

### 唯一约束

1. 唯一约束用来保证一列（或一组列）中的数据是唯一的
2. 与主键存在重要区别：

- 表可包含多个唯一约束，但每个表只允许一个主键
- 唯一约束列可包含NULL值
- 唯一约束列可修改或更新
- 唯一约束列的值可重复使用
- 与主键不一样，唯一约束列不能用来定义外键

1. UNIOUE约束：唯一约束既可以用UNIQUE关键字在表定义中定义，也可以用单独的CONSTRAINT定义

### 检查约束

1. 检查约束用来保证一列（或一组列）中的数据满足一组指定的条件
2. 常见用途：

- 检查最小或最大值。例如，防止0个物品的订单
- 指定范围。例如，保证发货日期大于等于今天的日期，但不超过今天起一年后的日期
- 只允许特定的值

1. 检查约束在数据类型内又做了进一步的限制，这些限制极其重要，可以确保插入数据库的数据正是你想要的数据。不需要依赖于客户端应用程序或用户来保证正确获取它，DBMS本身会拒绝任何无效的数据

```
CREATE TABLE OrderItems
(
  order_num   INTEGER   NOT NULL,
  order_item  INTEGER   NOT NULL,
  prod_id     CHAR(10)  NOT NULL,
  quantity    INTEGER   NOT NULL  CHECK(quantity > 0),
  item_price  MONEY     NOT NULL
);
```

1. 用户定义数据类型：定制数据类型的优点是只需施加约束一次（在数据类型定义中），而每当使用该数据时，都会自动应用这些约束。查阅相应的DBMS，看它是否支持自定义数据类型。

索引

1. 索引用来排序数据以加快搜索和排序操作的速度
2. 主键数据总是排序，这是DBMS的工作。按主键检索特定行总是一种快速有效的操作
3. 例子，想要搜索住在某个州的顾客。使用索引，在一个或多个列上定义索引，使DBMS保存其内容的一个排过序的列表。DBMS搜索排过序的索引，找出匹配位置，然后检索这些行
4. 创建索引前：

- 索引改善检索操作的性能，但降低了数据插入、修改和删除的性能。在执行这些操作时，DBMS必须动态地更新索引
- 索引数据可能要占用大量的存储空间
- 并非所有数据都合适做索引
- 索引用于数据过滤和数据排序
- 可以在索引中定义多个列

1. 创建索引：索引必须唯一命名。索引名在关键字之后定义。ON用来指定被索引的表，而索引中包含的列在表名后的圆括号中给出索引

```
CREATE INDEX prod_id_ind
ON Products (prod_name);
```

触发器

1. 触发器是特殊的存储过程，在特定的数据库活动发生时自动执行。触发器可与特定表上的INSERT、UPDATE和DELETE操作相关联
2. 触发器与单个的表相关联，与Orders表上的INSERT操作相关联的触发器只在Orders表中插入行时执行。类似地，Customers表上的INSERT和UPDATE操作的触发器只在表上出现这些操作时执行
3. 触发器内的代码具有以下数据的访问权：

- INSERT操作中的所有新数据
- UPDATE操作中的所有新数据和旧数据
- DELETE操作中删除的数据

1. 触发器的常见用途：

- 保持数据一致。例如，在INSERT或UPDATE操作中将所有州名转换为大写
- 基于某个表的变动在其他表上执行活动。例如，每当更新或删除一行时将审计跟踪记录写入某个日志表
- 进行额外的验证并根据需要回退数据
- 计算计算例的值或更新时间戳

1. 创建一个触发器，对所有INSERT和UPDATE操作，将Customers中的cust_state列转换为大写。这是SQL Server的版本

```
CREATE TRIGGER customer_state
ON Customers
FOR INSERT, UPDATE
AS
UPDATE Customers
SET cust_state = Upper(cust_state)
WHERE Customers.cust_id = inserted.cust_id;
```

**数据库安全**

1. DBMS给管理员提供管理机制，利用管理机制授予或限制对数据的访问
2. 任何安全系统的基础都是用户授权和身份确认。这是一种处理，通过这种处理对用户进行确认，保证他是有权用户，允许执行他要执行的操作
3. 需要保护的操作有：

- 对数据库的管理功能（创建表、更改或删除已存在的表等）的访问
- 对特定数据库或表的访问
- 访问的类型（只读、对特定列的访问等）
- 仅通过视图或存储过程对表进行访问
- 创建多层次的安全措施，从而允许多种基于登录的访问和控制
- 限制管理用户账号的能力

**小结：**

- 约束是实施引用完整性的重要部分
- 索引可改善数据检索的性能
- 触发器可以用来执行运行前后的处理
- 安全选项可用来管理数据访问