第 2 章　结构化查询语言简介

本章介绍结构化查询语言（SQL）。SQL 语句被分为两类：一类是数据操作语言（DML）语句，它被用来查询和修改数据，另一类是数据定义语言（DDL）语句，它被用来创建表、联系和其他的结构。本章只考虑用来查询数据的 DML 语句；其余的用来插入、修改和删除数据的 DML 语句将在第 7 章介绍。在第 7 章中，我们将同时介绍 SQL 的 DDL 语句。

2.1　SQL 的背景

SQL 在 20 世纪 70 年代后期由 IBM 公司开发，并于 1992 年被美国国家标准化协会（ANSI）认可为国家标准。本书所介绍的 SQL 版本基于这个标准，它有时候被称为 SQL-92。一个以后的版本，SQL3，结合一些面向对象的概念。这种较新的版本没有获得商业 DBMS 厂商的足够重视，因此对于实际的数据库处理来说不太重要。我们不在本书中讨论它。

不同于 Java 或是 C#，SQL 不是一种完整的编程语言。相反地，它被称为数据子语言，因为它只包括那些用来创建和处理数据库数据和元数据的语句。可以通过多种不同的方式来使用 SQL 语句。可以将它们直接提交给 DBMS 来处理；可以将 SQL 语句嵌入到客户机/服务器应用程序中；可以将它们嵌入到 Web 页面；可以将它们用于报表和数据抽取程序；同样也可以直接从 Visual Studio .NET 和其他开发工具中执行 SQL 语句。

SQL 到处存在，因此 SQL 编程是一项重要的技能。今天，所有的 DBMS 产品都处理 SQL。像在第 1 章中解释的那样，如果读者使用过 Microsoft Access，就已经使用过 SQL 了，即使没有意识到。每次处理一个表单，创建一个报表，或是运行一个查询，Access 都生成 SQL 语句，将其发送给 Access 内部的 DBMS 引擎 Jet。要进行更多的基础数据库处理，读者需要揭示出被 Access 所隐藏的 SQL。更进一步地，一旦你了解 SQL，相对于必须使用 Access 的图形化表单、按钮和其他的工具来创建查询，就会发现以 SQL 直接书写查询语句更为方便。

企业级的 DBMS 系统比如 Oracle，DB2，SQL Server 和 MySQL[①]需要了解 SQL。在这些产品中，所有的数据操作都是使用 SQL 来表示的。

2.2　Cape Codd 户外运动

Cape Codd 户外运动是一个根据真实的户外零售设备供应商所虚构的公司。Cape Codd 在遍及美国和加拿大的 15 个零售店铺中销售娱乐用途的户外设备。它同时通过 Internet 上的 Web 店面应用和邮件订单的方式销售商品。所有的零售销售都被存储在一个由 Oracle 管理的销售数据库中，如图 2.1 所示。

① MySQL 是一种开发源代码的 DBMS 产品，可以从 www.mysql.com 下载。在 Linux 环境下，它相当流行。我们会在第 14 章对其进行一些介绍。

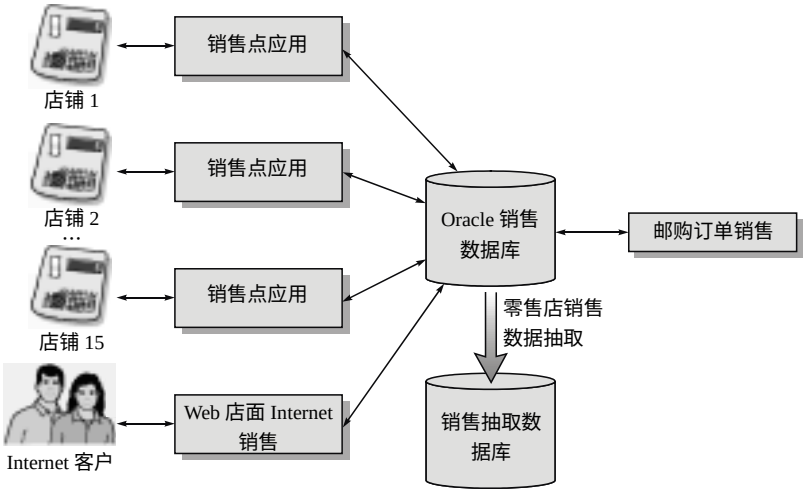


图 2.1　Cape Codd 零售销售抽取

2.2.1　零售数据抽取

Cape Codd 的市场部门打算对于店铺内的销售进行一次分析。相应地，市场分析人员要求信息服务部分从操作数据库中抽取零售销售数据。为进行市场研究，它们并不需要所有的订单数据。它们需要的表和列在图 2.2 中给出。

表	列	数据类型
RETAIL_ORDER	OrderNumber	Integer
	StoreNumber	Integer
	StoreZip	Character (9)
	OrderMonth	Character (12)
	OrderYear	Integer
	OrderTotal	Currency
ORDER_ITEM	OrderNumber	Integer
	SKU	Integer
	Quantity	Integer
	Price	Currency
	ExtendedPrice	Currency
SKU_DATA	SKU	Integer
	SKU_Description	Character (35)
	Department	Character (30)
	Buyer	Character (30)

图 2.2　零售销售抽取数据格式

需要三个表：RETAIL_ORDER，ORDER_ITEM 和 SKU_DATA。RETAIL_ORDER 表包含每个订单的数据，ORDER_ITEM 表包含订单中每个项目的数据，而 SKU_DATA 包含每个 SKU 的数据。这里 SKU 是库存单位的意思，读者可以认为 SKU 是每个 Cape Codd 公司所销售物品的标识。

2.2.2 RETAIL_ORDER 数据

如图 2.2 所示，RETAIL_ORDER 表有列 OrderNumber，StoreNumber，StoreZip（销售该订单的商铺邮政编码），OrderMonth，OrderYear 和 OrderTotal。我们只抽取有关零售店铺销售的数据。其他类型的销售（来回票据和其他销售相关的事务）都在抽取过程中被去掉。

数据抽取过程只选择操作数据的几个列。销售点（POS）和其他的应用所处理的数据比这里所给出的要多得多，同时它们保存数据的格式也不相同。例如，在 Oracle 数据库中，原始的订单数据以数据格式 MM/DD/YYYY 来存放 OrderDate（例如，10/22/2004 代表 2004 年 10 月 22 日）。抽取程序将 OrderDate 转化为市场部门需要的 OrderMonth 和 OrderYear 的格式。这种数据过滤和转换是数据抽取过程中常见的。

图 2.3 给出用于示例的 RETAIL_ORDER 数据。第一行包含 OrderNumber 1000 的数据抽取，第二行包含 OrderNumber 2000 的数据抽取，等等。

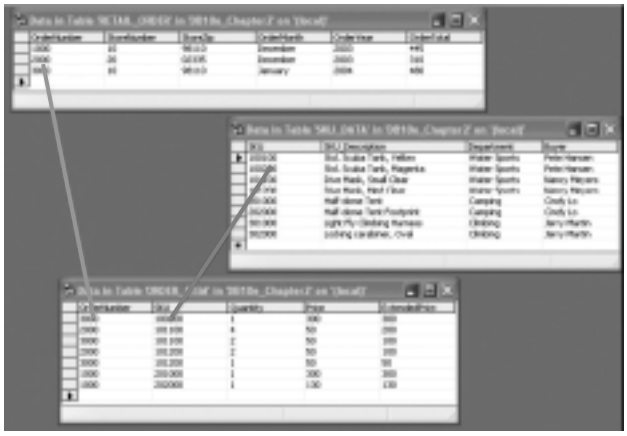


图 2.3 零售销售抽取的示例数据

2.2.3 ORDER_ITEM 数据

ORDER_ITEM 表包含每个订单中所购物品的数据抽取。对应于订单中的每个 SKU，表中都有相应的一行。要理解这个表，可以考虑一下你从零售商店中获得的销售回执，回执中包含每个订单的数据。它包含订单的基本数据，比如日期和总额，并且每一行对应你所购买的一种商品。ORDER_ITEM 表中的一行就对应类似订单回执中的条目。

OrderNumber 列和 RETAIL_ORDER 表中的 OrderNumber 列相关。SKU 是库存单位的编号，它和 SKU_DATA 表中的 SKU 相关（在下一节讨论）。Quantity 是该订单中这个 SKU 的购买数量。Price 是每个物品的价格，ExtendedPrice 是由 Quantity 和 Price 相乘得到的。图 2.3 的下部给出 ORDER_ITEM 数据。第一行和订单 3000，以及 SKU 100200 相关。订单 3000 中购买物品 100200，并且 ExtendedPrice 是 300。第二行和订单 2000 相关，在该行中，4 个物品 101100 以每个 50 美元的价格被购买，因此 ExtendedPrice 为 4 乘 50，即 200。这个结构对于订单中的物品是很典型的，我们会在第 5 章和第 6 章中再次看到它，在那里会为整个订单创建数据模型，并为这个数据模型设计数据库。

顺便提一下，读者可能认为在一个订单中，所有行的 ExtendedPrice 之和应该等于

RETAIL_ORDER 表中的 OrderTotal 列值。但是实际上它们并不相等。例如对于订单 3000，ExtendedPrice 之和等于 300 + 100 + 50，即 450。然而订单 3000 的 OrderTotal 值为 480。出现这个差别的原因在于，OrderTotal 中包含税、运输费和其他没有出现在数据抽取中的费用。

2.2.4　SKU_DATA 数据

SKU_DATA 表包含列 SKU，SKU_Description，Department 和 Buyer。SKU 是一个整型值，表示每一样 Cape Codd 出售的物品。SKU_Description 是一个对于每一样物品的简短文字描述。Department 和 Buyer 标识负责购买该物品的部门和个人。和其他的表一样，这些列也是操作数据库中所存储的 SKU 数据的一个子集。

2.2.5　数据抽取是普遍的

在继续学习之前，首先请读者注意，这里所介绍的数据抽取过程并不仅仅是一个理论化的练习。正相反，这种抽取过程是很现实、非常常见和重要的。目前，数以百计的全球商业公司正像 Cape Codd 一样在创建抽取数据库。

在本章的下一节，读者会学习如果编写 SQL 语句来处理这些抽取数据。这项知识非常有价值，并且很实用。再重复一下，就在你阅读这段话的时候，数以百计的人们正在编写 SQL 来从抽取数据中获得信息。在本章所学习的 SQL 将会是你作为知识工人、应用程序员或数据库管理员的重要资产。花时间来学习 SQL；这项投资会在你的职业中回报丰厚的利润。

2.3　SQL 的 SELECT/FROM/WHERE 框架

本节介绍 SQL 查询语句的基础语句框架。在我们讨论这个基础框架之后，读者会学习如何将 SQL 语句提交给 Access，SQL Server 和 Oracle 执行。如果愿意，可以像处理本章剩下部分所解释的 SQL 语句那样，随着教材的进程来学习额外的 SQL 语句。

2.3.1　从单一表中读取特定的列

从最简单开始。假设我们要获取 SKU_DATA 表的 Department 和 Buyer 列值。读取这些数据的 SQL 语句如下所示：

```
SELECT    Department, Buyer
FROM      SKU_DATA;
```

当 DBMS 为图 2.3 中的数据处理这条语句时，结果为：

Department	Buyer
Water Sports	Pete Hansen
Water Sports	Pete Hansen
Water Sports	Nancy Meyers
Water Sports	Nancy Meyers
Camping	Cindy Lo
Camping	Cindy Lo
Climbing	Jerry Martin
Climbing	Jerry Martin

SQL 语句转换表从表开始，以某种方式处理表，最后以表的结构提供结果。即使处理的结果

仅仅是一个单一的数字，这个数字也会被认为是一个一行一列的表。读者会在本章的后面学习到，一些 SQL 语句处理多个表。然而不管输入表的数目是多少，每条 SQL 语句的结果都是一个单一的输出表。

同时注意 SQL 语句以分号结尾，这是 SQL-92 标准所要求的。虽然一些 DBMS 产品允许忽略这个分号，但是另外一些有这个要求。因此请养成以分号结束 SQL 语句的习惯。

SELECT 语句中列名的顺序决定结果表中列的顺序。因此，如果在 SELECT 语句中交换 Buyer 和 Department 的顺序，它们也会在结果表中同样交换顺序。由此，SQL 语句：

```
SELECT    Buyer, Department
FROM      SKU_DATA;
```

会产生这样的结果表：

Buyer	Department
Pete Hansen	Water Sports
Pete Hansen	Water Sports
Nancy Meyers	Water Sports
Nancy Meyers	Water Sports
Cindy Lo	Camping
Cindy Lo	Camping
Jerry Martin	Climbing
Jerry Martin	Climbing

注意在结果中有一些行是重复的。例如第一行和第二行中的数据是相同的。下面我们使用 DISTINCT 关键字来消除重复：

```
SELECT    DISTINCT Buyer, Department
FROM      SKU_DATA;
```

这个语句的结果为：

Buyer	Department
Cindy Lo	Camping
Jerry Martin	Climbing
Nancy Meyers	Water Sports
Pete Hansen	Water Sports

所有的重复行都被消除。

顺便提一下，SQL 不自动去除重复行的原因是因为做这件事情很浪费时间。为确定行是否有重复，每一行都必须和其他行进行比较。如果在表中有 100 000 行，检查会占用很长的时间。因此默认情况下，并不删除重复。然而总是可以使用 DISTINCT 关键字来强迫删除重复行。

假如我们要查看 SKU_DATA 表的所有列。为此我们可以在 SELECT 语句中指定每个列，如下所示：

```
SELECT    SKU, SKU_Description, Department, Buyer
FROM      SKU_DATA;
```

结果将会是包含 SKU_Data 中所有行和所有 4 个列的表。然而，SQL 提供一种简洁的符号来查询一个表的所有列，这种方式是使用一个星号：

```
SELECT    *
FROM      SKU_DATA;
```

结果为：

SKU	SKU_Description	Department	Buyer
100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo
301000	Light Fly Climbing Harness	Climbing	Jerry Martin
302000	Locking carabiner, Oval	Climbing	Jerry Martin

它包含 SKU_DATA 表中的所有行和所有列数据。

2.3.2　从单一表中读取特定的行

假如我们要选择 SKU_DATA 表的所有列，而只要与水上运动部门相关的行。可以以如下的方式来使用 WHERE 子句：

```
SELECT    *
FROM      SKU_DATA
WHERE     Department = 'Water Sports';
```

这个 SQL 语句的结果为：

SKU	SKU_Description	Department	Buyer
100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

在一个 WHERE 子句中，如果列包含字符或是日期类型，用于比较的值必须以单引号（' '）引起来。如果这个列包含的是数字类型数据，则比较的值不需要在引号内。因此，要寻找所有 SKU 值大于 200 000 的行，我们可以这样写：

```
SELECT    *
FROM      SKU_DATA
WHERE     SKU > 200000;
```

结果为：

SKU	SKU_Description	Department	Buyer
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo
301000	Light Fly Climbing Harness	Climbing	Jerry Martin
302000	Locking carabiner, Oval	Climbing	Jerry Martin

如同前面所说明的那样，对于数字类型列值的比较不需要被括在括号内。同时注意在数字类型值中，不使用逗号。

2.3.3 从单一表中读取特定的列和行

到目前为止，我们已经可以选择特定列和所有行，以及所有列和特定行了。我们可以将这些操作结合在一起，通过为需要的列命名和使用 WHERE 子句来选择特定列及特定行。例如，要选择登山部门中所有产品的 SKU_Description 和部门名称，可指定：

```
SELECT    SKU_Description, Department
FROM      SKU_DATA
WHERE     Department = 'Climbing';
```

结果为：

SKU_Description	Department
Light Fly Climbing Harness	Climbing
Locking carabiner, Oval	Climbing

SQL 并不需要在 WHERE 子句中使用的列也同样出现在 SELECT 的列名列表中。因此，可以指定：

```
SELECT    SKU_Description, Buyer
FROM      SKU_DATA
WHERE     Department = 'Climbing';
```

在这里，限定列 Department 并没有出现在 SELECT 的列名列表中。结果为：

SKU_Description	Buyer
Light Fly Climbing Harness	Jerry Martin
Locking carabiner, Oval	Jerry Martin

顺便提一下，标准练习中我们将 SQL 语句的 SELECT，FROM 和 WHERE 都写在单独的行上。这只是一种编码惯例，事实上 SQL 解析器并不要求这样。你可以这样书写最后一条 SQL 语句，SELECT SKU_Description，Buyer FROM SKU_DATA WHERE Department = 'Climbing'；所有的内容都在一行上，任何 DBMS 产品仍然可以处理它。然而标准的多行 SQL 编码规则使得 SQL 更容易阅读。我们鼓励读者按照这个标准来书写自己的 SQL。

2.3.4 向 DBMS 提交 SQL 语句

在继续介绍 SQL 之前，学习如何向 DBMS 提交 SQL 语句是很有用的。通过这种方式，读者可以在学习本书时，一边输入和运行 SQL 语句，一边阅读对其的讨论。

顺便提一下，即使不在 DBMS 上运行查询，也可以学习 SQL 语句。如果出于某种原因，读者并没有 Access，SQL Server 或是 Oracle，不要绝望；你可以在没有它们的情况下学习 SQL。情况可能是，你的教师和现在实践中的绝大多数人一样，都是在没有 DBMS 的情况下学习 SQL 的。只是如果可以在阅读时同时运行 SQL，则会更容易理解和记住 SQL 语句。

具体提交 SQL 语句的方法取决于 DBMS。这里我们将介绍在 Access，SQL Server 和 Oracle 中提交 SQL 的情况。

将 **SQL** 提交给 **Access**

在可以执行 SQL 语句之前，需要一台已经安装 Access 的计算机，并且需要包含图 2.3 中的

表和示例数据的 Access 数据库。Access 是许多版本的 Microsoft Office 中的一部分，因此要找到一台安装有它的计算机并不困难。

一些菜单选项在不同版本的 Access 中有细微的差别。这里的讨论是基于 Access 2003。如果读者拥有的是不同版本的 Access，命令可能有细微的差别。然而这并不是什么大问题。

读者有两种方式可以获得图 2.3 中的数据库、表和数据。首先，你可以从本书的 Web 站点 www.prenhall.com/kroenke 下载名为 Chapter_2.mdb 的 Access 数据库。此外，你也可以使用附录 A 中所说明的过程来自己创建数据库，并添加图 2.3 中的表和数据。在继续以前，需要采取其中的一种方式。

为在 Access 中处理 SQL 语句，打开数据库，并点击 Access 窗口左上角的查询标签来创建一个新的查询窗口。然后在数据库窗口的顶部点击 New 选项，如图 2.4 所示。点击 OK 按钮来选择设计视图，并点击显示表对话框的关闭（×）按钮（因为我们要自己输入 SQL 语句，因此不必在这个窗口中指定表）。接下来，在 Access 菜单中选择 View 选项，并点击 SQL View 选项，如图 2.5 所示。

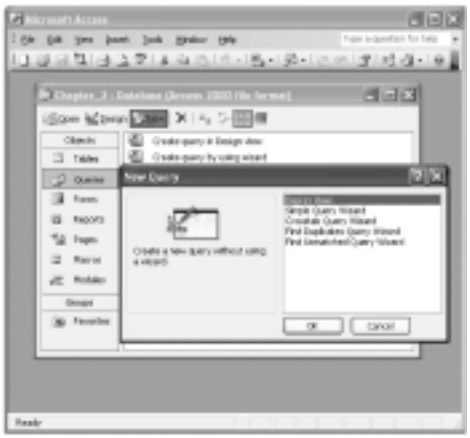


图 2.4　在 Access 中创建一个新的查询

图 2.5　在 Access 中选择 SQL View 选项

接着，在出现的空白窗口中输入 SQL 语句。如图 2.6 所示，输入如下的 SQL 语句：

```
SELECT    *
FROM      SKU_DATA;
```

在输入 SQL 语句后，在 Access 菜单中点击 Query 选项，然后像图 2.6 一样点击 Run 选项。结果如图 2.7 所示。

图 2.6　在 Access 中运行 SQL 代码

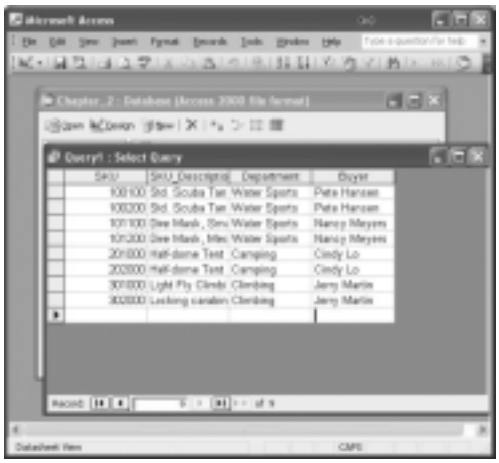


图 2.7　在 Access 中执行查询的结果

如果要修改查询语句，或是键入新的内容，在 Access 菜单中选择 View 选项，并和前面一样选择 SQL View 选项。在这里，可以修改 SQL 语句，或是键入一条新语句。也可以在查询窗口激活时，通过选择 File/Save 选项来存储这条语句。

顺便提一下，Microsoft Access 是一个针对初学者的 DBMS 产品。它并不能正确处理本章所介绍的全部 SQL 语句。而这里所展示的都是读者应该掌握的标准 SQL 语句。我们不将讨论仅仅限定于 Access 可以处理的语句，而是介绍所有重要的 SQL 语句，并标记那些 Access 不能处理的东西（在 Access 中无效）。Access 不能处理某些 SQL 语句是很令人烦心的。如果这让读者感到困扰，写信给 Microsoft，告诉他们去修改 Access 的 SQL 解析器。

将 **SQL** 提交给 **SQL Server**

在能够为 SQL Server 输入 SQL 语句之前，读者应该可以访问一台装有 SQL Server 的计算机，并有包含图 2.3 中给出的表和数据的数据库。你的教师可能已经在计算机实验室中安装了 SQL

Server，并为你输入数据。如果是这样的话，按照他的指示来访问数据库。

另一种选择是，如果你购买与本书捆绑销售的 SQL Server 试用版[①]，可以将该试用版安装在任意运行 Windows 的计算机上，并创建图 2.3 中的表和数据。你需要阅读第 11 章中关于 SQL Server 的介绍。同样地，本书的 Web 站点包含可以帮助你在 SQL Server 下创建 Cape Codd 抽取数据库的文件和指令。

最简单的在 SQL Server 中练习 SQL 的办法是在查询分析器中输入 SQL 语句，我们将在第 11 章中对此进行说明。图 2.8 给出如下 SQL 语句在 SQL Server 查询分析器中的执行：

```
SELECT    *
FROM      SKU_DATA;
```

将 **SQL** 提交给 **Oracle** 数据库

在能够为 Oracle 输入 SQL 语句之前，读者应该可以访问一台装有 Oracle 的计算机，并有包含图 2.3 中给出的表和数据的数据库。

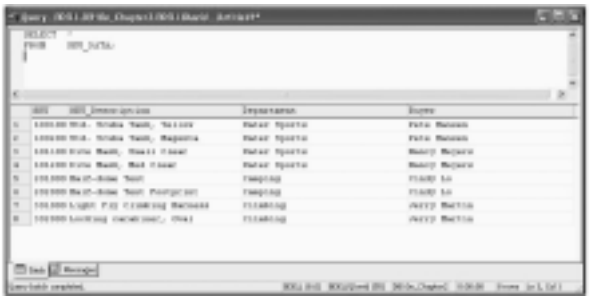


图 2.8　使用 SQL Server 的查询分析器执行 SQL

你的教师可能已经在计算机实验室中安装了 Oracle，并为你输入数据。如果是这样的话，按照他的指示来访问数据库。另一种选择是，如果你购买与本书捆绑销售的 Personal Oracle 的试用版，可以将该试用版安装在任意运行 Windows 的计算机上，并创建图 2.3 中的表和数据。我们将在第 10 章开始对 Oracle 进行介绍，可以参照那里的指令进行。同样地，本书的 Web 站点包含可以帮助你在 Oracle 下创建 Cape Codd 抽取数据库的文件和指令。

最简单的在 Oracle 中练习 SQL 的办法是在 Oracle 的一个工具 SQL Plus 中输入 SQL 语句，我们将在 10 章中对此进行介绍。图 2.9 给出如下 SQL 语句在 SQL Plus 的执行情况：

```
SELECT    SKU, DEPARTMENT, BUYER
FROM      SKU_DATA;
```

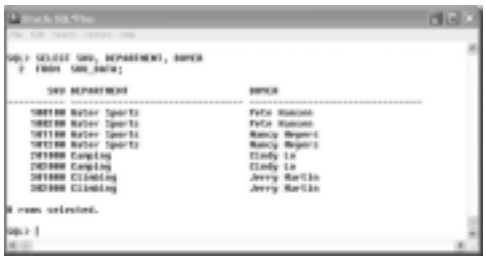


图 2.9　使用 Oracle 的 SQL Plus 执行 SQL

① 仅限于在美国的读者——编者注。

读者也可以在 Oracle 中使用图形化 Windows 工具来输入 SQL 语句，我们在 10.2.6 节对此进行介绍。然而这样会使读者脱离真正数据库管理员所使用的经典 Oracle 环境。如果你打算为 Oracle 工作，请至少使用一段时间的 SQL Plus。

2.4 查询单一表的 SQL

本节介绍更多的处理单一表的 SQL 语句。随着本书的进展，读者会发现 SQL 对于查询数据库和从已有的数据中产生的信息是多么强大。作为图示，这里的输出由 SQL Server 生成，其他 DBMS 产品的输出也是类似的。

2.4.1 将结果排序

SQL 语句产生的行序是任意的，这是由每个 DBMS 内部的程序所决定的。图 2.3 中给出的例子结果是由 Access 产生的，其顺序也由 Access 确定。Oracle，SQL Server，DB2 和其他的 DBMS 产品会产生不同的顺序。同时注意在 ORDER_ITEM 表中，对应订单 3000 的行是分散在结果中的。

如果你要求 DBMS 以特定的顺序显示行，可以使用词语 ORDER BY。例如，下面的 SQL 语句：

```
SELECT    *
FROM      ORDER_ITEM
ORDER BY  OrderNumber;
```

将会有如下的结果：

OrderNumber	SKU	Quantity	Price	ExtendedPrice
1000	201000	1	300.0000	300.0000
1000	202000	1	130.0000	130.0000
2000	101100	4	50.0000	200.0000
2000	101200	2	50.0000	100.0000
3000	101200	1	50.0000	50.0000
3000	101100	2	50.0000	100.0000
3000	100200	1	300.0000	300.0000

通过添加第二个列名，我们可以在两个列上排序。例如，为首先使用 OrderNumber 排序，然后在同一个 OrderNumber 内使用 Price 排序，我们这样写：

```
SELECT    *
FROM      ORDER_ITEM
ORDER BY  OrderNumber, Price;
```

结果为：

OrderNumber	SKU	Quantity	Price	ExtendedPrice
1000	202000	1	130.0000	130.0000
1000	201000	1	300.0000	300.0000
2000	101100	4	50.0000	200.0000
2000	101200	2	50.0000	100.0000
3000	101200	1	50.0000	50.0000
3000	101100	2	50.0000	100.0000
3000	100200	1	300.0000	300.0000

如果希望首先使用 Price，然后再用 OrderNumber 对数据排序，应该如下交换 ORDER BY 子句后列的顺序：

```
SELECT    *
FROM      ORDER_ITEM
ORDER BY  Price, OrderNumber;
```

对 Access 用户的说明：不同于 SQL Server 这里的输出情况，Access 在现金数据的输出时添加了美元符号。

默认情况下，行按照升序排列。为按照降序排列，在列名后面添加关键字 DESC。因此，为首先按照 Price 降序排列，再按照 OrderNumber 升序排列，可以指定：

```
SELECT    *
FROM      ORDER_ITEM
ORDER BY  Price DESC, OrderNumber ASC;
```

结果为：

OrderNumber	SKU	Quantity	Price	ExtendedPrice
1000	201000	1	300.0000	300.0000
3000	100200	1	300.0000	300.0000
1000	202000	1	130.0000	130.0000
2000	101100	4	50.0000	200.0000
2000	101200	2	50.0000	100.0000
3000	101200	1	50.0000	50.0000
3000	101100	2	50.0000	100.0000

由于默认的顺序是升序，我们不需要在最后的 SQL 语句中指明 ASC。因此，下面的 SQL 语句是等价的：

```
SELECT    *
FROM      ORDER_ITEM
ORDER BY  Price DESC, OrderNumber;
```

2.4.2　WHERE 子句选项

SQL 包含一组 WHERE 子句的选项，这可以极大地扩展 SQL 的功能和应用。在本节中，我们考虑三个选项：复合子句、范围和通配符。

复合 WHERE 子句

SQL 的 WHERE 子句可以使用 AND，OR，IN 和 NOT IN 运算符来包含多个条件。例如，要选择 SKU_DATA 中所有部门名称为水上运动，而买主为 Nancy Meyers，我们可以写为：

```
SELECT    *
FROM      SKU_DATA
WHERE     Department = 'Water Sports'
  AND     Buyer = 'Nancy Meyers';
```

结果为：

SKU	SKU_Description	Department	Buyer
101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

类似地，要选择 SKU_DATA 中或者属于野营部门，或者属于登山部门的数据，可以编写：

```
SELECT    *
FROM      SKU_DATA
WHERE     Department = 'Camping'
   OR     Department = 'Climbing';
```

结果为：

SKU	SKU_Description	Department	Buyer
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo
301000	Light Fly Climbing Harness	Climbing	Jerry Martin
302000	Locking carabiner, Oval	Climbing	Jerry Martin

三个或者更多的 AND 和 OR 条件可以组合使用，但在这种情况下，使用 IN 和 NOT IN 运算符会更容易。例如，假设要获取 SKU_DATA 中所有买主为 Nancy Meyers，Cindy Lo 或 Jerry Martin 其中之一的行，我们可以构建一个包含两个 AND 的 WHERE 子句。但另一种简单的途径是如下面这样使用 IN 关键字：

```
SELECT    *
FROM      SKU_DATA
WHERE     Buyer IN ('Nancy Meyers', 'Cindy Lo', 'Jerry Martin');
```

在这种格式中，一个值的集合被括在括号里。如果买主等于其中任意一个值，则所在的行被选中。结果为：

SKU	SKU_Description	Department	Buyer
101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo
301000	Light Fly Climbing Harness	Climbing	Jerry Martin
302000	Locking carabiner, Oval	Climbing	Jerry Martin

类似地，如果我们要在 SKU_DATA 中寻找买主不是 Nancy Meyers，Cindy Lo 或 Jerry Martin 中任何一位，可以这样写：

```
SELECT    *
FROM      SKU_DATA
WHERE     Buyer NOT IN ('Nancy Meyers', 'Cindy Lo', 'Jerry Martin');
```

结果为：

SKU	SKU_Description	Department	Buyer
100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen

注意 IN 和 NOT IN 的重要区别。一行满足 IN 的条件（如果它等于括号内的任意一个值）。而一行满足 NOT IN 的条件（如果它不等于括号中的所有值）。

WHERE 子句中的范围

SQL 的 WHERE 子句可以使用 BETWEEN 关键字来指定数据值的范围。例如，下面的 SQL

语句：

```
SELECT    *
FROM      ORDER_ITEM
WHERE     ExtendedPrice BETWEEN 100 AND 200;
```

会产生如下的结果：

OrderNumber	SKU	Quantity	Price	ExtendedPrice
2000	101100	4	50.0000	200.0000
3000	101100	2	50.0000	100.0000
2000	101200	2	50.0000	100.0000
1000	202000	1	130.0000	130.0000

注意范围的上界和下界，即 100 和 200，也被包含在结果表中。前面的 SQL 语句相当于：

```
SELECT    *
FROM      ORDER_ITEM
WHERE     ExtendedPrice > = 100
  AND     ExtendedPrice < = 200;
```

同时注意，ORDER BY 关键字可以和任意 WHERE 子句组合使用。

```
SELECT    *
FROM      ORDER_ITEM
WHERE     ExtendedPrice BETWEEN 100 AND 200
ORDER BY  OrderNumber DESC;
```

它将会产生这样的结果：

OrderNumber	SKU	Quantity	Price	ExtendedPrice
3000	101100	2	50.0000	100.0000
2000	101200	2	50.0000	100.0000
2000	101100	4	50.0000	200.0000
1000	202000	1	130.0000	130.0000

WHERE 子句中的通配符

可以在 WHERE 子句中使用关键字 LIKE 来指定对于列值的部分匹配。例如，假设要在 SKU_DATA 表中寻找所有买主名为 Pete 的行。为实现这个目的，我们使用如下带有通配符%的关键字 LIKE：

```
SELECT    *
FROM      SKU_DATA
WHERE     Buyer LIKE 'Pete%';
```

百分号（%）是一个代表任意字符序列的通配符。当它和 LIKE 一起使用时，字符串'Pete%'意味着所有以单词 Pete 开头的字符串。这个查询的结果是：

SKU	SKU_Description	Department	Buyer
100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen

顺便提一下，通配符%和下划线（_）字符是在 SQL-92 标准中定义的。它们可以使用在除 Microsoft 的 Access 以外的所有 DBMS 产品中。对于 Access，使用星号（*）代替%，

使用问号（?）代替下划线。这个差异之所以存在，是由于 Access 的设计人员选择遵循 Microsoft DOS 下的通配符，而不是 SQL-92 的通配符。

假设我们要寻找在 SKU_DATA 的 SKU_Description 里，描述中的某个地方包含单词 Tent 的行。由于 Tent 可以出现在开始、结尾或者中间，我们需要在 LIKE 短语的两端都加上通配符，如下所示：

```
SELECT    *
FROM      SKU_DATA
WHERE     SKU_Description LIKE '%Tent%';
```

结果为：

SKU	SKU_Description	Department	Buyer
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo

这个查询会找到在 SKU_Description 中的任意地方出现单词 Tent 的行。

有些时候我们需要在列的某个特定位置寻找某个特定的值。例如，假定在 SKU 值的编码中，从右侧开始的第三个位置上的 2 有特殊的含义，比如表明它是其他商品的变种。不管出于什么原因，假定要寻找从右侧开始的第三个位置上是 2 的所有 SKU。假如我们使用如下的 SQL 语句：

```
SELECT    *
FROM      SKU_DATA
WHERE     SKU LIKE '%2%';
```

结果为：

SKU	SKU_Description	Department	Buyer
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
201000	Half-dome Tent	Camping	Cindy Lo
202000	Half-dome Tent Footprint	Camping	Cindy Lo
302000	Locking carabiner, Oval	Climbing	Jerry Martin

这不是所要的结果，我们错误地选择了所有在 SKU 值中任何地方有 2 的行。

为准确地寻找到商品，我们不能使用 SKU LIKE'%2%'的表示。不同的是，我们必须使用下划线（_）来表示单个、不确定的字符。下面的 SQL 语句会找到所有的 SKU_DATA 行，它在右侧第三个位置上是 2：

```
SELECT    *
FROM      SKU_DATA
WHERE     SKU LIKE '%2_ _';
```

注意到这里有两个下划线，一个代表右边的第一个位置，而另一个代表右边的第二个位置。结果为：

SKU	SKU_Description	Department	Buyer
100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

这就是我们想要的结果。

2.5　在 SQL 查询中进行计算

可以在 SQL 查询语句中进行一些类型的算术运算。一类计算涉及到内置的 SQL 函数的使用，另一类涉及到在 SELECT 语句中的列上进行简单的算法操作。我们依次考虑它们。

2.5.1　使用 SQL 内建的函数

SQL 提供 5 个内置的函数用于在表的列上进行算术运算：SUM，AVG，MIN，MAX 和 COUNT。一些 DBMS 产品扩展这些标准的内置函数来提供一些额外的函数。这里，我们重点讨论这 5 个标准函数。

假设要知道 RETAIL_ORDER 表中所有订单的 OrderTotal 之和，可以采用如下的方法来获得这个总数：

```
SELECT    SUM (OrderTotal)
FROM      RETAIL_ORDER;
```

结果为：

(No column name)
1235.0000

回想一下，SQL 语句的结果总是一个表。在这里例子中，该表只包含一行和一列，其值为 OrderTotal 之和。

OrderTotal 的和并不是表中的一列，因此 DBMS 不能为它提供列名。前述的结果来自于 SQL Server，它将该列命名为'（No column name）'。其他 DBMS 产品所存取的举措也是类似的。

这个结果看上去不太好。我们希望有一个有意义的列名，SQL 允许我们使用 AS 关键字为其设置一个列名。如果我们指定：

```
SELECT    SUM (OrderTotal) AS OrderSum
FROM      RETAIL_ORDER;
```

结果为：

OrderSum
1235.0000

这是一个更有意义的标签。这里的名字 OrderSum 是任意的，可以自由地选择我们认为对于用户来说该结果有意义的名字。可以选择 OrderTotal_Total，OrderTotalSum 或者任意其他的什么。

如果将内置函数用于 WHERE 子句，则可以增强其效用。例如，我们可以书写：

```
SELECT    SUM (ExtendedPrice) AS Order3000Sum
FROM      ORDER_ITEM
WHERE     OrderNumber = 3000;
```

结果为：

Order3000Sum
450.0000

内置的函数可以在一个语句中混合和匹配，例如，我们可以书写：

```
SELECT    SUM (ExtendedPrice) AS OrderItemSum,
          AVG (ExtendedPrice) AS OrderItemAvg,
          MIN (ExtendedPrice) AS OrderItemMin,
          MAX (ExtendedPrice) AS OrderItemMax
FROM      ORDER_ITEM;
```

结果为：

OrderItemSum	OrderItemAvg	OrderItemMin	OrderItemMax
1160.0000	165.5714	50.0000	300.0000

函数 COUNT 看上去和 SUM 类似，但实际上是不同的。COUNT 计算行的数目，而 SUM 累加列的值。如果我们书写：

```
SELECT    COUNT(*) AS NumRows
FROM      ORDER_ITEM;
```

结果为：

NumRows
7

这个结果表明表中共有 7 行。注意如果要计算行数的话，需要在 COUNT 函数后加上一个星号。COUNT 是惟一一个需要星号的内置函数。此外，COUNT 可以被应用于任意类型的数据，而 SUM，AVG，MIN 和 MAX 只能被应用于数值型数据。

COUNT 产生的结果在某些情况下可能令人意外。例如，假设要计算 SKU_DATA 表中部门的数目。如果写为：

```
SELECT    COUNT (Department) AS DeptCount
FROM      SKU_DATA;
```

结果为：

DeptCount
8

这是 SKU_DATA 表中行的数目，而不是不同部门值的数目。如果你要计算不同部门值的数目，则需要使用如下的 DISTINCT 关键字：

```
SELECT    COUNT (DISTINCT Department) AS DeptCount
FROM      SKU_DATA;
```

{在 Access 中失效}

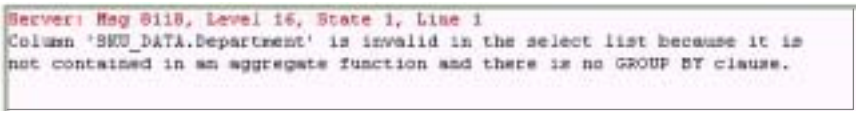
结果为：

DeptCount
3

除分组（在后面定义）以外，不可以将表的列名和一个内置函数结合在一起。如果编写：

```
SELECT    Department, COUNT(*)
FROM      SKU_DATA;
```

在 SQL Server 中的结果为：



（这个出错信息是 SQL Server 专有的。然而在 Access，Oracle 或 DB2 中，也会看到相类似的信息）。

需要了解内置函数的其他限制，它们不可以使用于 WHERE 子句。因此，不能这样写：

```
SELECT    *
FROM      RETAIL_ORDER
WHERE     OrderTotal>AVG(OrderTotal);
```

试图这样编写语句也会导致来自于 DBMS 的错误信息。在第 7 章中，读者会学习如何使用一系列的 SQL 视图来获取想要的结果。

2.5.2　SELECT 语句中的算术运算

可以在 SQL 语句中进行基本的算术运算。例如假设为验证 ORDER_ITEM 表中数据的准确性，要计算总价的值。我们这样书写 SQL 语句来计算总价：

```
SELECT    Quantity * Price AS EP
FROM      ORDER_ITEM;
```

结果为：

EP
300.0000
200.0000
100.0000
100.0000
50.0000
300.0000
130.0000

如果我们打算将这个计算得到的值和原来存储的 ExtendedPrice 值进行比较，可以这样写：

```
SELECT    Quantity * Price AS EP, ExtendedPrice
FROM      ORDER_ITEM;
```

结果为：

EP	ExtendedPrice
300.0000	300.0000
200.0000	200.0000
100.0000	100.0000
100.0000	100.0000
50.0000	50.0000
300.0000	300.0000
130.0000	130.0000

现在我们可以可视化地比较这两个值，以确保存储的值是准确的。

另一个 SQL 语句中表达式的用途是进行字符串操作。假设我们要将 Buyer 和 Department 列结合成一个单一的名为 Sponsor 的列。如果这样写这个语句：

```
SELECT    Buyer + 'in' + Department AS Sponsor
```

```
FROM      SKU_DATA;
```

结果为：

Sponsor	
Pete Hansen	in Water Sports
Pete Hansen	in Water Sports
Nancy Meyers	in Water Sports
Nancy Meyers	in Water Sports
Cindy Lo	in Camping
Cindy Lo	in Camping
Jerry Martin	in Climbing
Jerry Martin	in Climbing

结果的形式不太好。我们可以使用更高级的函数来去除空白。其相关语法和使用方法在不同的 DBMS 中是不同的。如果我们在这里对每个 DBMS 产品的不同特性进行说明，则会偏离讨论的重点。如果读者要了解更多的信息，在所使用的 DBMS 文档中搜索有关字符串函数的部分。

仅仅为说明这种可能性，我们这里给出有关 SQL Server 的语句，消除 Buyer 和 Department 右侧结尾的空白：

```
SELECT    DISTINCT RTRIM (Buyer) + 'in' + RTRIM (Department) AS Sponsor
FROM      SKU_DATA;
```

结果为：

Sponsor
Cindy Lo in Camping
Jerry Martin in Climbing
Nancy Meyers in Water Sports
Pete Hansen in Water Sports

这个结果在视觉上要好一些。

2.6 分组

在 SQL 中，可以使用 GROUP BY 关键字来对行依照相同的值进行分组。例如，假设在 SKU_DATA 表上的 SELECT 语句上指定 GROUP BY Department，DBMS 会首先按照部门把所有的行排序，然后将所有有相同部门值的行归成一组。针对于每一个不同的部门值，都会有相应的一个组。

例如，SQL 语句：

```
SELECT     Department, COUNT(*) AS Dept_SKU_Count
FROM       SKU_DATA
GROUP BY   Department;
```

产生如下的结果：

Department	Dept_SKU_Count
Camping	2
Climbing	2
Water Sports	4

为获取这个结果，DBMS 首先依照部门的值对行进行排序，然后计算具有相同部门值的行数。

另一个使用 GROUP BY 的例子是：

```
SELECT    SKU, AVG (ExtendedPrice) AS AvgEP
FROM      ORDER_ITEM
GROUP BY  SKU;
```

该语句结果为：

SKU	AvgEP
100200	300.0000
101100	150.0000
101200	75.0000
201000	300.0000
202000	130.0000

这里行被按照 SKU 进行排序和分组，然后计算每组 SKU 物品的平均 ExtendedPrice。

可以在 GROUP BY 表达式中使用不止一个列。例如，SQL 语句：

```
SELECT    Department, Buyer, COUNT(*) AS Dept_Buyer_SKU_Count
FROM      SKU_DATA
GROUP BY  Department, Buyer;
```

首先按照部门，再按照买主的值，对行进行分组；然后计算每个部门值和买主值组合所对应的行数。其结果为：

Department	Buyer	Dept_Buyer_SKU_Count
Camping	Cindy Lo	2
Climbing	Jerry Martin	2
Water Sports	Nancy Meyers	2
Water Sports	Pete Hansen	2

当使用 GROUP BY 时，只有在 GROUP BY 表达式中出现的列和 SQL 的内置函数可以被应用于 SELECT 表达式。下面的表示会产生错误：

```
SELECT    SKU,Department,COUNT(*) AS Dept_SKU_Count
FROM      SKU_DATA
GROUP BY  Department;
```

这样的语句是错误的，因为对应每个部门组，会有多个不同的 SKU 值。DBMS 没有办法将多个值放置在结果中。如果读者不理解这个问题，可以试着手工来处理这个语句，它是不能进行的。

当然，WHERE 和 ORDER BY 子句也可以在 SELECT 语句中使用，如下所示：

```
SELECT    Department, COUNT(*) AS Dept_SKU_Count
FROM      SKU_DATA
WHERE     SKU <> 302000
GROUP BY  Department
ORDER BY  Dept_SKU_Count;
```

{在 Access 中失效}

结果为：

Department	Dept_SKU_Count
Climbing	1
Camping	2
Water Sports	4

注意登山部门信息的一行被从记数中删除了，因为它不满足 WHERE 条件。如果不使用

ORDER BY 子句，则行会以任意的部门顺序显示。当使用它以后，就可以给出顺序。

通常情况下，将 WHERE 放置在 GROUP BY 之前。一些 DBMS 产品不要求这个顺序，但另一些有这个要求。为安全起见，总是将 WHERE 放在 GROUP BY 的前面。

SQL 提供另外一个 GROUP BY 的特性，以扩展其功能。HAVING 运算符限制在结果中出现的组。我们可以限制前面的查询，只显示包含有超过一行的组，其写法如下：

```
SELECT    Department, COUNT(*) AS Dept_SKU_Count
FROM      SKU_DATA
WHERE     SKU <> 302000
GROUP BY  Department
HAVING    COUNT (*) > 1
ORDER BY  Dept_SKU_Count;
```

{在 Access 中无效}

结果为：

Department	Dept_SKU_Count
Camping	2
Water Sports	4

将这个结果与前面的结果比较，对应于攀登的一行（数量为 1）被去除。

顺便提一下，任何 SQL 的内置函数都可以用于 HAVING 子句。例如，下面是一个有效的 SQL：

```
SELECT    COUNT(*) AS SKU_Count, SUM(Price) AS TotalRev, SKU
FROM      ORDER_ITEM
GROUP BY  SKU
HAVING    SUM (Price) = 100;
```

运行这个查询来查看结果。

注意当语句中同时包含 WHERE 和 HAVING 子句时，可能有一些含糊的情况。最终的结果取决于 WHERE 条件在 HAVING 之前还是之后应用。为消除这种含糊，WHERE 总是在 HAVING 之前被应用。

2.7 在 NASDAQ 交易数据中寻找模式

在继续对于 SQL 的讨论之前，考虑一个例子，来展示我们刚刚介绍的 SQL 的能力。

假设一个朋友告诉你，她怀疑股票市场倾向于在一周的某些特定日子上涨，而在其他的日子下跌。她请求你去检查过往的交易数据来判断这是否是事实。特别是她希望交易一个称为 NASDAQ 100 的指数基金，该基金是一个针对 NASDAQ 交易最大的 100 家公司的交易所基金。她提供你过去年份中 NASDAQ 100 的交易数据以供分析。假定她提供的数据以关系数据库中一个名为 NDX 表的形式存储（可以在本书的网站 www.prenhall.com/kroenke 找到这个表）。

2.7.1 检查数据的特性

假定你决定首先检查数据的一般特性。通过执行下面的查询来查看这个表中存在的列：

```
SELECT    *
FROM      NDX;
```

该查询结果的前 5 行数据如下：

TClose	PriorClose	ChangeClose	Volume	TMonth	TDayOfMonth	TYear	TDayOfWeek	TQuarter
1520.46	1530.65	-10.1900...	24827600.0	January	9	2004	Friday	1
1530...	1514.26	16.39000...	26839500.0	January	8	2004	Thursday	1
1514.26	1501.26	13.0	22962800.0	January	7	2004	Wednesday	1
1501.26	1496.58	4.680000...	22732200.0	January	6	2004	Tuesday	1
1496...	1463.57	33.00999...	23629100.0	January	5	2004	Monday	1

假设你知道第一行是该基金在某个交易日休市时的值，而第二行则是前一个交易日休市时的值，第三行是当天休市值和前一天休市值的差。Volume 是所有交易的数目，而其他的数据都和交易日期相关。

下面，你决定使用这个查询语句来检查交易价格的变动：

```
SELECT    AVG (ChangeClose) AS AverageChange,
          MAX (ChangeClose) AS MaxGain,
          MIN (ChangeClose) AS MaxLoss
FROM      NDX;
```

结果为：

AverageChange	MaxGain	MaxLoss
0.28116703819958445	399.59999999999991	-401.02999999999992

顺便提一下，DBMS 提供很多函数来格式化查询结果，以减少小数点后数字的显示位数，为结果添加现金符号，比如$或£，或者进行其他的格式修改。然而这些函数是和 DBMS 相关的。使用术语 formatting results 搜索你所使用 DBMS 的文档，来了解更多有关这些函数的信息。

仅仅是出于好奇，你决定寻找那些有最大和最小变动的日子。为避免在长数字串中寻找位置，以进行相等的比较，使用大于和小于来对相近的值进行比较：

```
SELECT    ChangeClose, TMonth, TDayOfMonth, TYear
FROM      NDX
WHERE     ChangeClose > 398
   OR     ChangeClose < -400;
```

结果为：

ChangeClose	TMonth	TDayOfMonth	TYear
-401.02999999999993	January	3	1994
399.59999999999991	January	3	2001

结果是令人吃惊的！有什么理由最大的涨幅和最大的跌幅都出现在 1 月 3 日吗？你开始怀疑是否你的朋友有一个有希望的想法。

2.7.2　在一周的日交易中寻找模式

希望确定是否在一周的日平均交易中存在着差异。相应地，创建如下的 SQL 语句：

```
SELECT    TDayOfWeek, AVG (ChangeClose) AS AvgChange
FROM      NDX
GROUP BY  TDayOfWeek;
```

结果为：

TDayOfWeek	AvgChange
Monday	-1.0357792946529938
Tuesday	-0.7114406779660856
Friday	0.14602173913045174
Wednesday	0.7779405520170050?
Thursday	2.1741297297297497

确实，在一周的不同交易日上似乎存在着差异。NASDAQ 100 似乎在周一和周二下跌，而在其他的三天中上涨。特别是周四，似乎是一个非常好的交易日。

然而，你开始怀疑，是否该模式在每年都成立呢？为回答这个问题，编写 SQL 语句：

```
SELECT    TDayOfWeek, TYear, AVG (ChangeClose) AS AvgChange
FROM      NDX
GROUP BY  TDayOfWeek, TYear
ORDER BY  TDayOfWeek, TYear DESC;
```

因为有 20 年的数据，这个查询结果共有 100 行。为简化分析，决定将行限定为最近的 5 年：

```
SELECT    TDayOfWeek, TYear, AVG (ChangeClose) AS AvgChange
FROM      NDX
WHERE     TYear > '1999'
GROUP BY  TDayOfWeek, TYear
ORDER BY  TDayOfWeek, TYear DESC;
```

这个查询的部分结果如下：

TDayOfWeek	TYear	AvgChange
Friday	2004	-7.27000000000000955
Friday	2003	-2.4849999999999635
Friday	2002	-2.1941999999999688
Friday	2001	-19.594399999999979
Friday	2000	8.898839215686296?
Monday	2004	33.009999999999991
Monday	2003	3.7741666666666802
Monday	2002	-2.6022916666666447
Monday	2001	-3.7527083333332976
Monday	2000	-19.899574468085014
Thursday	2004	16.390000000000001
Thursday	2003	5.7070000000000229
Thursday	2002	-3.779799999999975
Thursday	2001	9.3144000000000329
Thursday	2000	24.766274509803957
Tuesday	2004	4.6800000000000637
Tuesday	2003	4.4130769230769431
Tuesday	2002	-7.858823529411759
Tuesday	2001	-8.8845999999999723
Tuesday	2000	-3.5062745098038532
Wednesday	2004	13.0
Wednesday	2003	-1.6959615384615176

至少在这段时间对于这个基金而言，看来一周中的某一天并不是一个很好的对于上涨或是下跌的预言者。

我们会继续讨论，以分析这些数据，但到目前为止你应该了解到 SQL 对于处理表是非常有用的。本章的最后给出一些推荐的额外 SQL 习题。

现在我们介绍查询两个或多个表的 SQL 语句，作为本章的结束。

2.8　使用 SQL 查询两个或多个表

假定读者要了解由水上运动部门所管理的 SKU 所产生的收入，这个收入可以通过对 ExtendedPrice 求和来得到，但是我们会遇到一个问题。ExtendedPrice 存放在表 ORDER_ITEM，而部门信息则存放在表 SKU_DATA。我们需要处理两个表中的数据，而目前为止所介绍的 SQL 语句都只能一次在一个表上工作。

SQL 提供两种不同的技术来从多个表中查询数据：子查询和联接。虽然这两种技术都可以在多个表上工作，但它们在用途上有细微的差别。我们将在下面对其进行介绍。

2.8.1　使用子查询查询多个表

我们如何获得由水上运动部门所管理的物品所产生的 ExtendedPrice 之和呢？如果知道这些物品的 SKU 值，则我们可以使用一个带有 IN 关键字的 WHERE 子句。

对于图 2.3 中的数据，水上运动部门的物品 SKU 值是 100100，100200，101100 和 101200。知道这些值，我们可以使用下面的 SQL 语句来获得它们的 ExtendedPrice 之和：

```
SELECT    SUM (ExtendedPrice) AS Revenue
FROM      ORDER_ITEM
WHERE     SKU IN (100100, 100200, 101100, 101200);
```

结果为：

Revenue
750.0000

但在通常情况下，我们并不预先知道需要的 SKU 值。然而有一种途径可以获取这些值，通过在 SKU_DATA 表上使用 SQL 语句。为获得水上运动部门的 SKU 值，只需要书写：

```
SELECT    SKU
FROM      SKU_DATA
WHERE     Department = 'Water Sports';
```

该 SQL 语句的结果为：

SKU
100100
100200
101100
101200

这就是我们所要的 SKU 值。

现在需要将最后两个 SQL 语句结合在一起，以获得想要的结果。我们将第一个 SQL 语句中的列表值替换为第二个 SQL 表达式，结果如下：

```
SELECT    SUM (ExtendedPrice) AS Revenue
FROM      ORDER_ITEM
WHERE     SKU IN
          (SELECT   SKU
          FROM      SKU_DATA
          WHERE     Department = 'Water Sports');
```

其结果为：

Revenue
750.0000

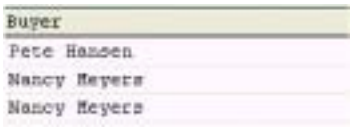
这与之前的结果是相同。

第二个被包含在括号中的 SELECT 语句被称为子查询。我们可以使用多个子查询来处理三个甚至是更多的表。例如，假设想知道那些在 2004 年 1 月购买商品的买主姓名。首先，注意到买主的信息被存储在表 SKU_DATA，而 OrderMonth 和 OrderYear 则存储在表 RETAIL_ORDER。

我们可以使用一个包含两个子查询的 SQL 语句来获得想要的数据，SQL 语句如下：

```
SELECT    Buyer
FROM      SKU_DATA
WHERE     SKU IN
          (SELECT   SKU
          FROM      ORDER_ITEM
          WHERE     OrderNumber IN
                    (SELECT   OrderNumber
                    FROM      RETAIL_ORDER
                    WHERE     OrderMonth = 'January'
                    AND       OrderYear = 2004));
```

该语句的结果为：



Buyer
Pete Hansen
Nancy Meyers
Nancy Meyers

为理解这个语句，我们自底向上查看。底端的 SELECT 语句获取在 2004 年 1 月出售订单的 OrderNumbers 列表。中间的 SQL 语句获取那些在 2004 年 1 月出售订单中物品的 SKU 值。最后，最高层的 SELECT 语句获得那些在中间层 SELECT 语句中找到的 SKU 的买主。

读者在本章前面所学到的所有 SQL 语句都可以被应用到子查询所获得的表上，不管这些 SQL 看上去有多么复杂。例如，我们可以在结果上应用 DISTINCT 来消除重复行。或者，可以像下面这些应用 GROUP BY 和 ORDER BY：

```
SELECT    Buyer, COUNT (*) AS NumberSold
FROM      SKU_DATA
WHERE     SKU IN
          (SELECT   SKU
          FROM      ORDER_ITEM
          WHERE     OrderNumber IN
                    (SELECT   OrderNumber
                    FROM      RETAIL_ORDER
                    WHERE     OrderMonth = 'January'
                    AND       OrderYear = 2004))
GROUP BY Buyer
ORDER BY NumberSold DESC;
```

{在 Access 中无效}

结果为：

Buyer	NumberSold
Nancy Meyers	2
Pete Hansen	1

子查询是非常强大的，但它们有严重的限制。被选中的数据只能来自于最高层的表。不能使用子查询来获取来自于超过一个表的数据。为达到这个目的，必须使用联接来作为替代。

2.8.2　使用联接查询多个表

联接（join）操作用来结合两个或多个表，这是通过将一个表中的行和其他表中的行连接（concatenating）（黏在一起）来实现的。考虑图 2.10 中的两个表。表 STUDENT 有关于学生信息的 3 个列，而表 PROJECT_　GRADE 有关于学生上课成绩信息的 3 个列。

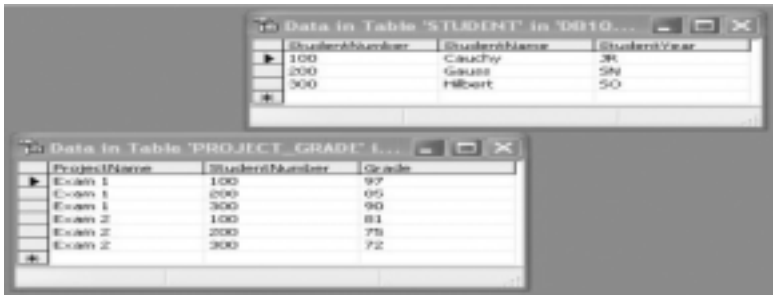


图 2.10　STUDENT 和 PROJECT_GRADE 数据

我们可以使用下面的 SQL 语句来将一个表中的行和第二个表中的行连接起来：

```
SELECT    *
FROM      STUDENT, PROJECT_GRADE;
```

这个语句会将第一个表中的每一行和第二个表中的每一行黏在一起，就图 2.10 中的数据而言，其结果为：

StudentNumber	StudentName	StudentYear	ProjectName	StudentNumber	Grade
100	Cauchy	JR	Exam 1	100	97
100	Cauchy	JR	Exam 1	200	85
100	Cauchy	JR	Exam 1	300	90
100	Cauchy	JR	Exam 2	100	81
100	Cauchy	JR	Exam 2	200	75
100	Cauchy	JR	Exam 2	300	72
200	Gauss	SN	Exam 1	100	97
200	Gauss	SN	Exam 1	200	85
200	Gauss	SN	Exam 1	300	90
200	Gauss	SN	Exam 2	100	81
200	Gauss	SN	Exam 2	200	75
200	Gauss	SN	Exam 2	300	72
300	Hilbert	SO	Exam 1	100	97
300	Hilbert	SO	Exam 1	200	85
300	Hilbert	SO	Exam 1	300	90
300	Hilbert	SO	Exam 2	100	81
300	Hilbert	SO	Exam 2	200	75
300	Hilbert	SO	Exam 2	300	72

由于有 3 行的学生信息和 6 行的考试信息，因此在这个表中有 3×6，即 18 行。注意到学生 Cauchy 和 PROJECT_GRADE 表中的全部 6 行相连接，学生 Gauss 和 Hilbert 的情况也是相同的。

如果现在读者是这三个学生之一，那就一定会对这个表有反对意见。你不但和自己的成绩关联在一起，还和其他学生的成绩也关联在一起。这不符合逻辑；我们需要做的是将那些在表 STUDENT 和表 PROJECT_GRADE 之间 StudentNumber 相等的行连接起来。这很容易实现，我们只需要对这个 SQL 语句应用 WHERE 子句：

```
SELECT    *
FROM      STUDENT, PROJECT_GRADE
WHERE     STUDENT.StudentNumber = PROJECT_GRADE.StudentNumber;
```

结果为：

StudentNumber	StudentName	StudentYear	ProjectName	StudentNumber	Grade
100	Cauchy	JR	Exam 1	100	97
200	Gauss	SR	Exam 1	200	85
300	Hilbert	SO	Exam 1	300	90
100	Cauchy	JR	Exam 2	100	81
200	Gauss	SR	Exam 2	200	75
300	Hilbert	SO	Exam 2	300	72

如果我们将这个结果和图 2.10 中的数据比较，会发现只有适当的成绩才和学生进行关联。同时发现我们可以达到这个目的。因为在结果中，STUDENT（第一列）的每一行的 StudentNumber 都等于 PROJECT_GRADE（倒数第二列）的 StudentNumber 值。这在前面的结果中并不成立。

顺便提一下，语法 STUDENT.StudentNumber 表示的是表 STUDENT 中的 StudentNumber。同样，PROJECT_GRADE.StudentNumber 指的是 PROJECT_GRADE 表中的 StudentNumber。读者总是可以通过这样的表名来限定列名。在前面我们没有这样做，是因为当时只在一个表上操作。然而在使用比如 SKU_DATA.Buyer 的语法，而不是仅仅 Buyer，或是 ORDER_ITEM.Price，而不是 Price 后，前面介绍的 SQL 语句仍然有效。

通过连接（concatenating）两个表得到的表称为联接（join），创建这个表的过程称为联接两个表。当表被使用一个相等的条件（例如在 StudentNumber 的情况）联接时，这个联接称为同等联接。当人们提到联接时，99.99999%的时候都是指的同等联接。

我们可以使用联接来从两个或多个表中获取数据。例如，使用图 2.3 中的数据，假设要查询买主的姓名和这个买主所购买的全部销售的 ExtendedPrice，可以使用下面的 SQL 语句：

```
SELECT    Buyer, ExtendedPrice
FROM      SKU_DATA, ORDER_ITEM
WHERE     SKU_DATA.SKU = ORDER_ITEM.SKU;
```

结果为：

Buyer	ExtendedPrice
Pete Hansen	300.0000
Nancy Meyers	200.0000
Nancy Meyers	100.0000
Nancy Meyers	100.0000
Nancy Meyers	50.0000
Cindy Lo	300.0000
Cindy Lo	130.0000

同样地，每个 SQL 语句的结果只是一个表，所以我们可以将所学习的所有在单个表上的 SQL 语句应用到这个结果上。例如，我们可以如下地使用 GROUP BY 和 ORDER BY：

```
SELECT    Buyer, SUM(ExtendedPrice) AS BuyerRevenue
FROM      SKU_DATA, ORDER_ITEM
WHERE     SKU_DATA.SKU = ORDER_ITEM.SKU
GROUP BY  Buyer
ORDER BY  BuyerRevenue DESC;
```

{在 Access 中无效}

结果为：

Buyer	BuyerRevenue
Nancy Meyers	450.0000
Cindy Lo	430.0000
Pete Hansen	300.0000

我们可以扩展这个语法来联接三个或更多的表。例如，假设要获取买主和每个买主所购买全部物品的 ExtendedPrice 和 OrderMonth，我们需要联接三个表来获取这些数据。方式如下：

```
SELECT    Buyer, ExtendedPrice, OrderMonth
FROM      SKU_DATA, ORDER_ITEM, RETAIL_ORDER
WHERE     SKU_DATA.SKU = ORDER_ITEM.SKU
  AND     ORDER_ITEM.OrderNumber = RETAIL_ORDER.OrderNumber;
```

结果为：

Buyer	ExtendedPrice	OrderMonth
Pete Hansen	300.0000	January
Nancy Meyers	200.0000	December
Nancy Meyers	100.0000	January
Nancy Meyers	100.0000	December
Nancy Meyers	50.0000	January
Cindy Lo	300.0000	December
Cindy Lo	130.0000	December

同样可以通过使用 ORDER BY 或者分组来改进这个查询。

联接也可以使用其他的语法来书写，读者还需要了解一些在值缺失时的联接问题，但不在本章对此进行介绍，我们将会在第 7 章中对联接进行讨论。

2.8.3　比较子查询和联接

子查询和联接都处理多个表，但是它们有细微的差别。如前所述，子查询只可以被用来从顶层表中获取数据。而联接可以从任意数目的表中获取数据。因此，联接可以做任何子查询可以做的工作，并且可以做得更多。那为什么我们还要学习子查询呢？一方面，如果只要从单一表中查询数据，则使用子查询更为容易书写，也更容易理解。这在处理多个表时尤其明显。

然而在第 8 章中，将会学习另一种类型的子查询，称为有相关子查询。这类子查询可以做联接所不能做的工作。因此，同时学习联接和子查询是非常重要的，即使目前看来似乎联接一律都比较优越。如果读者好奇，雄心勃勃，并且富有勇气，那么可以阅读 8.2 节对于相关子查询的讨论。

2.9 小结

喔，这是完整的一章！

有两种 SQL 语句：DML 和 DDL 语句。DML 包含用于查询数据和插入、修改以及删除数据的语句。本章只介绍用于查询的 DML 语句。

SQL 是由 IBM 开发的，并被 ANSI SQL-92 确认为一项标准。SQL 是一种数据子语言，它可以被嵌入到其他的完整程序设计语言或是直接提交给 DBMS 处理。了解 SQL 对于知识工人、应用开发人员和数据库管理员都是非常重要的。

所有的 DBMS 都处理 SQL。Access 将 SQL 隐藏起来，而 SQL Server，Oracle，DB2 和 MySQL 要求你使用 SQL。

本章中的例子基于从 Cape Codd 运动公司的操作数据库中抽取出的三个表。类似的数据抽取很常见，同时也很重要。图 2.3 给出这三个表的一些示例数据。

SQL 查询语句的基本结构是 SELECT/FROM/WHERE。在 SELECT 后列出要选择的列，在 FROM 后给出要处理的表，而在 WHERE 后给出所有对于数据值的约束。在一个 WHERE 子句中，字符和日期数据必须以单引号引起来，而数值型数据则不需要。像本章所描述的那样，可以直接将 SQL 语句提交给 Access，SQL Server 和 Oracle。

本章介绍以下一些关键字的使用：ORDER BY，DESC，ASC，AND，OR，IN，NOT IN，BETWEEN，LIKE，%［对于 Access 是星号（*）］，_［对于 Access 是问号（?）］，SUM，AVG，MIN，MAX，COUNT，AS，GROUP BY 和 HAVING。读者应该知道如何通过混合和匹配这些特性来取得所要的结果。默认情况下，WHERE 子句在 HAVING 子句前先执行。

读者可以使用子查询和联接来检索多个表。子查询是使用关键字 IN 和 NOT IN 的嵌套查询。一个 SQL 的 SELECT 表达式被放置在括号内。使用子查询，可以只显示顶层表的数据。联接是通过在 FROM 子句中指定多个表名来实现的，WHERE 子句则被用来获取同等联接。在大多数情况下，同等联接是最直观的选择。联接可以显示来自于多个表的数据。在第 8 章中，读者将会学到另一种类型的子查询，它不能通过联接来实现。

2.10 习题

2.1 SQL 代表什么？

2.2 DML 代表什么？什么是 DML 语句？

2.3 DDL 代表什么？什么是 DDL 语句？

2.4 概述 SQL 的背景。

2.5 SQL-92 是什么？它和本章中介绍的 SQL 语句有什么关系？

2.6 为什么 SQL 被描述为一种数据子语言？

2.7 描述三种使用 SQL 的方式。

2.8 说明 Access 如何使用 SQL。

2.9 说明企业级的 DBMS 产品如何使用 SQL。

2.10 SKU 代表什么？什么是一个 SKU?

2.11 概述在 Cape Codd 数据抽取中，数据是如何被修改和过滤的。

2.12 大致说明 RETAIL_ORDER，ORDER_ITEM 和 SKU_DATA 表之间的联系。

使用下面的表来回答习题 **2.13** 至习题 **2.40** 的问题：

```
INVENTORY (SKU, Description, QuantityOnHand, QuantityOnOrder,
Warehouse)
```

假定 Description 和 Buyer 包含字符数据，而其他的列包含数字数据。如果读者有 Access 系统，在本书附带的数据库 Chapter_2.mdb 中，表 INVENTORY 上运行你的查询。这个数据库可以从本书的 Web 站点 www.prenhall.com/kroenke 下载。

2.13 书写 SQL 语句来显示 SKU 和 Description。

2.14 书写 SQL 语句来显示 Description 和 SKU。

2.15 书写 SQL 语句来显示 Warehouse。

2.16 书写 SQL 语句来显示不重复的 Warehouse。

2.17 不使用*，书写 SQL 语句来显示所有的列。

2.18 使用*，书写 SQL 语句来显示所有的列。

2.19 书写 SQL 语句来显示所有商品数据，只要其 QuantityOnHand 大于 0。

2.20 书写 SQL 语句来显示商品的 SKU 和 Description，只要其 QuantityOnHand 等于 0。

2.21 书写 SQL 语句来显示商品的 SKU，Description 和 Warehouse，要求 QuantityOnHand 等于 0，且将结果按照 Warehouse 升序排列。

2.22 书写 SQL 语句来显示商品的 SKU，Description 和 Warehouse，要求 QuantityOnHand 等于 0，且以 Warehouse 的降序和 QuantityOnHand 的升序排列。

2.23 书写 SQL 语句来显示商品的 SKU 和 Description，要求 QuantityOnHand 等于 0，且 QuantityOnOrder 大于 0。

2.24 书写 SQL 语句来显示商品的 SKU 和 Description，要求 QuantityOnHand 等于 0，或者 QuantityOnOrder 等于 0。

2.25 书写 SQL 语句来显示商品的 SKU 和 Description，要求它们被存储在 Seattle，Chicago 或者 New Jersey 的仓库，不要使用 IN 关键字。

2.26 书写 SQL 语句来显示商品的 SKU 和 Description，要求它们被存储在 Seattle，Chicago 或者 New Jersey 的仓库，使用 IN 关键字。

2.27 书写 SQL 语句来显示商品的 SKU 和 Description，要求它们被存储在 Seattle，Chicago 或者 New Jersey 的仓库，不要使用 NOT IN 关键字。

2.28 书写 SQL 语句来显示商品的 SKU 和 Description，要求它们被存储在 Seattle，Chicago 或者 New Jersey 的仓库，使用 NOT IN 关键字。

2.29 书写 SQL 语句来显示商品的 SKU，Description 和 QuantityOnHand，要求 QuantityOnHand 大于 1，且小于 10，使用 BETWEEN。

2.30 书写 SQL 语句来显示商品的 SKU 和 Description，要求 description 以'Half-dome'开头。

2.31 书写 SQL 语句来显示商品的 SKU 和 Description，要求 description 包含单词'Foot'。

2.32 书写 SQL 语句来显示商品的 SKU 和 Warehouse，要求在 Warehouse 列中，左起第三个字母是'w'。

2.33 书写 SQL 表达式，在 QuantityOnHand 列上使用所有的内置函数。在结果中使用有意义的列名。

2.34 说明 COUNT 和 SUM 的不同。

2.35 书写 SQL 语句来生成一个单一的名为 ItemLocation 列，它包含 Description，短语"is located in,"和 Warehouse 的组合，要求商品的 QuantityOnHand 大于 0。不考虑关于去除空白的问题。

2.36 书写 SQL 语句来显示 Warehouse 和按照 Warehouse 分组的 QuantityOnHand 数目，将该数目所在的列命名为 TotalItemsOnHand，并按照 TotalItemsOnHand 的降序排列。

2.37 书写 SQL 语句来显示 Warehouse 和按照 Warehouse 分组的 QuantityOnHand 数目，忽略所有数目大于 2 的。将该数目所在的列命名为 TotalItemsOnHand，并按照 TotalItemsOnHand 的降序排列。

2.38 书写 SQL 语句来显示 Warehouse 和按照 Warehouse 分组的 QuantityOnHand 数目。只显示所有数目小于 2 的。将该数目所在的列命名为 TotalItemsOnHand，并按照 TotalItemsOnHand 的降序排列。

2.39 在对习题 2.38 的回答中，是 WHERE 子句还是 HAVING 子句先做？为什么?

2.40 书写 SQL 语句来显示 Warehouse，按照 Warehouse 和 QuantityOnOrder 分组的 QuantityOnOrder 数目和 QuantityOnHand 数目。忽略所有超过 2 的数目。将 QuantityOnOrder 的数目列命名为 TotalItemsOnOrder，QuantityOnHand 的数目列命名为 TotalItemsOnHand。

使用下面的表来回答习题 **2.41** 至习题 **2.47**：

```
INVENTORY (SKU, Description, QuantityOnHand, QuantityOnOrder,
Warehouse)
```
WAREHOUSE (Warehouse, Manager, SquareFeet)

2.41 书写 SQL 语句来显示物品的 SKU 和 Description，要求它们被存放在由'Smith'管理的仓库中。使用子查询。

2.42 书写 SQL 语句来显示物品的 SKU 和 Description，要求它们被存放在由'Smith'管理的仓库中。使用联接。

2.43 书写 SQL 语句来显示 Warehouse 和该 Warehouse 中 QuantityOnHand 的平均值，要求该仓库由'Smith'管理。使用子查询。

2.44 书写 SQL 语句来显示 Warehouse 和该 Warehouse 中 QuantityOnHand 的平均值，要求该仓库由'Smith'管理。使用联接。

2.45 书写 SQL 语句来显示所有物品的 Warehouse，Manager 和 QuantityOnHand，要求该仓库由'Smith'管理。使用联接。

2.46 说明为什么在习题 2.45 的答案中，不能使用子查询。

2.47 说明子查询和联接有什么不同。

项目练习

下面的问题使用 2.7.1 节的 NDX 数据表。读者可以从本书的 Web 站点 www.prenhall/kroenke 找到该数据。它是一个 Access 的数据库（Chapter_2.mdb）。

2.48 书写 SQL 查询来获取以下的结果：

a．星期五的 ChangeClose。

b．星期五的最小、最大和平均 ChangeClose。

c．按照 TYear 分组的平均 ChangeClose，同时显示 TYear。

d．按照 TYear 和 TMonth 分组的平均 ChangeClose，同时显示 TYear 和 TMonth。

e．按照 TYear，TQuarter 和 TMonth 分组的平均 ChangeClose，要求按照平均值的降序排列（需要为该平均值赋予一个名称，以便按照它排序）。同时显示 TYear，TQuarter 和 TMonth。注意月份是依字母序而不是日历序的。请说明需要怎么做才可以获得日历序的月份。

f．按照 TYear，TQuarter 和 TMonth 分组的最大 ChangeClose 和最小 ChangeClose 之间的差值，要求按照降序排列（需要为该差值赋予一个名称，以便按照它排序）。同时显示 TYear，TQuarter 和 TMonth。

g．按照 TYear 分组的平均 ChangeClose，依降序排列（需要为该平均值赋予一个名称，以便按照它排序）。要求只显示平均值为正的分组。

h．以年/月/日的形式，显示一个单一的日期字段。不考虑结尾空白的情况。

2.49　交易量（交易的份额数）可能与证券市场的买卖方向有关。使用在本章学习的 SQL 来研究这种可能性，并至少给出 5 个不同的 SQL 语句。

Marcia 干洗店项目练习

Marcia 干洗店是一个迎合高消费层次者的干洗店，它位于富裕的郊区附近。通过提供高级的客户服务，Marcia 干洗店使其业务表现显著领先于其他竞争者。Marcia 干洗店希望可以追踪每个客户及其订单；并最终通过电子邮件的方式通知客户的衣服已经洗好。为提供这项服务，干洗店开发初始的带有几张表的数据库。其中三张表如下：

```
CUSTOMER (Phone, FirstName, LastName, Email)
INVOICE (Number, DateIn, DateOut, TotalAmt, Phone)
INVOICE_ITEM (Number, Item, Quantity, UnitPrice)
```

编写 SQL 语句来生成如下的信息：

A　显示每张表中的所有数据。

B　列出每个客户的电话号码和姓。

C　列出所有名为'Nikki'客户的电话号码和姓。

D　列出所有超出 100 元订单的电话号码、开始时间和结束时间。

E　列出所有名字以'B'开始的客户的电话号码和名字。

F　列出所有姓包含字符'cat'的客户的电话号码和名字。

G　列出所有电话号码第二位和第三位分别为 2 和 3 的客户的电话号码和姓名。

H　确定最大和最小的总价。

I　确定平均的总价。

J　计算客户数。

K　按照 LastName 和 FirstName 对客户分组。

L　计算不同名同姓的客户数。

M　使用子查询，给出拥有单一订单总价超过 100 元的客户姓名。结果先按照姓升序排列，再名降序排列。

N　使用联接，给出拥有单一订单总价超过 100 元的客户姓名。结果先按照姓升序排列，再按照名降序排列。

O　使用子查询，给出拥有包含物品'Dress Shirt'的订单的客户姓名。结果先按照姓升序排列，再按照名降序排列。

P　使用联接，给出拥有包含物品'Dress Shirt'的订单的客户姓名。结果先按照姓升序排列，再按照名降序排列。

Q　使用带有子查询的联接，给出拥有包含物品'Dress Shirt'的订单的客户姓名和总价。结果先按照姓升序排列，再按照名降序排列。

Morgan 进口公司项目练习

Morgan 进口公司从亚洲购买古玩和家庭装饰品，然后将它们船运到位于 Los Angeles 的一个仓库。

Morgan 先生使用一个数据库来维护购买物品的列表，装船情况和船运的物品。他的数据库包含以下的表：

```
SHIPMENT (Number, Shipper, DepartureDate, ArrivalDate, InsuredValue)
SHIPMENT_ITEM (Number, Item, Quantity, Value)
ITEM_PURCHASE (Item, Store, Quantity, City, Date, LocalCurrencyAmt,
ExchangeRate)
```

编写 **SQL** 语句来得到以下的信息：

A 显示表中的所有数据。

B 列出所有船运的编号和发货人。

C 列出所有保险金额超过 10 000 元船运的编号和发货人。

D 列出所有名字以'AB'开头船运的编号和发货人。

E 假设 DepartureDate 和 ArrivalDate 的格式为月/日/年。列出所有于 12 月出发船运的编号、发货人和到达日期。

F 假设 DepartureDate 和 ArrivalDate 的格式为月/日/年。列出所有于某月 10 日出发船运的编号、发货人和到达日期。

G 确定最大和最小的保险金额。

H 确定平均保险金额。

I 计算船运的总数。

J 对于表 ITEM_PURCHASE 中每一行，显示物品、商店和一个计算得到的名为 StdCurrency Amount 的列。该列的值为 LocalCurrencyAmt 乘上 ExchangeRate。

K 按照城市和商店对购买的物品分组。

L 按照城市和商店对购买的物品分组，计算每组的数目。

M 使用子查询，显示所有包含单价超过 1000 物品船运的托运人和出发日期。结果先按照托运人升序排列，再按照出发日期降序排列。

N 使用联接，显示所有包含单价超过 1000 物品船运的托运人和出发日期。结果先按照托运人升序排列，再按照出发日期降序排列。

O 使用子查询，显示所有包含在新加坡购买物品船运的托运人和出发日期。结果先按照托运人升序排列，再按照出发日期降序排列。

P 使用联接，显示所有包含在新加坡购买物品船运的托运人和出发日期。结果先按照托运人升序排列，再按照出发日期降序排列。

Q 使用子查询和联接的组合，显示船运的托运人和出发日期，以及在新加坡购买物品的价格。结果先按照托运人升序排列，再按照出发日期降序排列。