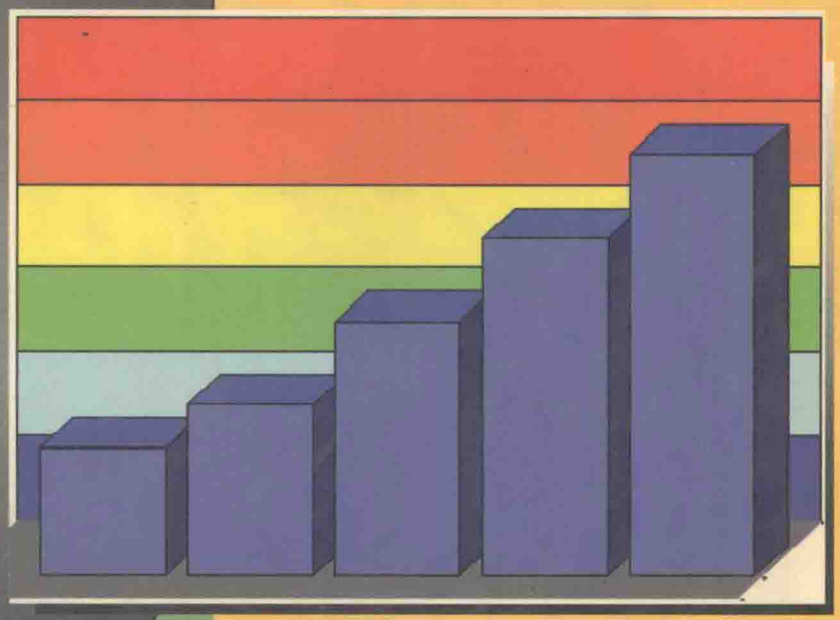
计算机最新技术培训教材

PEKING UNIVERSITY PRESS

杨浩广 编著

Visual C++ 6.0 数据库开发学习教程



- 详细讲述Visual C++ 6.0 数据库开发的方方面面
- 从入门到精通的Visual C++ 6.0 数据库开发教程

北京大学出版社
http://cbs.pku.edu.cn

计算机最新技术培训教材

Visual C++ 6.0
数据库开发学习教程

杨浩广　编著

北京大学出版社
北　京

内 容 简 介

本书是给那些使用 Visual C++的数据库开发人员一个进一步提高数据库开发能力台阶的教程。Visual C++中的数据库编程有四个方面：通过开放数据库连接(ODBC)接口调用，通过 ODBC 的 MFC 数据库类，通过 DAO 调用，通过 OLE DB 和 ADO 调用。本书针对以上四个方面，分别讲述了它们的基础知识和如何编写它们的数据库应用程序，并在书中讲述了大量数据库编程实例和技巧。

图书在版编目(CIP)数据

Visual C++ 6.0 数据库开发学习教程/杨浩广编著．—北京：北京大学出版社，2000.5
ISBN 7-301-01731-6

Ⅰ.V… Ⅱ.杨… Ⅲ.C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2000)第 07003 号

书　　名：Visual C++ 6.0 数据库开发学习教程
著作责任者：杨浩广
责任编辑：杨锡林
标准书号：ISBN 7-301-01731-6/TP·111
出 版 者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
网　　址：http://cbs.pku.edu.cn
电　　话：出版部 62752015　发行部 62754140　编辑室 62765013
电子信箱：xxjs@pup.pku.edu.cn
排 版 者：兴盛达激光照排中心
印 刷 者：河北省滦县印刷厂
发 行 者：北京大学出版社
经 销 者：新华书店
787 毫米×1092 毫米　16 开本　16.75 印张　415 千字
2000 年 5 月第一版　2000 年 5 月第一次印刷
定　　价：27.00 元

前　言

数据库的应用极为广泛，对于一个 C++编程人员来说，数据库编程就成为一个重要的主题。

特别是 Visual C++6.0,它是在 Windows 95/98 和 Windows NT 上建立 32 位数据库应用程序的强大工具。它为数据库开发人员提供了方便的数据库开发接口。

本书将给那些希望进一步提高数据库开发能力的 Visual C++6.0 开发人员一个台阶。为了使读者对 Visual C++6.0 的数据库开发的基本原理有一个全面的了解，首先讲述一些数据库的基础概念。

Visual C++6.0 中的数据库编程有四个方面：通过开放数据库连接(ODBC)接口调用，通过 ODBC 的 MFC 数据库类，通过 DAO 调用，通过 OLE DB 和 ADO 调用。

微软与许多软件公司共同引入了一个标准，用来帮助数据库开发人员寻求一种简单而功能强大的方式，通过这种方式在他们的应用程序中支持多个 DBMS。这个方式就是 ODBC(Open Database Connectivity,开放式数据库互连)。与 Visual C++6.0 一起提供的 ODBC 库，允许应用程序与许多不同的数据库连接，它提供了一个普通的编程接口，以使用范围很广的各种数据库。

尽管 ODBC 确实有其优点，但即使进行最简单的数据库操作也要使用大量的代码。令人欣慰的是，从 MFC 的 1.5 版本开始，引入了一组封装 ODBC 功能的 MFC 类，这些类通常被称为 MFC 数据库类。

ODBC 胜过其他数据库技术的优点之一是允许一个单代码具有与各种数据库接口的能力。很不幸，这导致了“最小共同特性”的代码，这样就不能利用每个数据库特有的功能。

微软利用 DAO3.0 解决了这个问题，DAO3.0 通过 Automation(自动化，以前称为 OLE Automation)提供 Jet 引擎的所有功能。对于任何 Automation 类型库，一个客户应用程序可以通过 OLE Idispatch 接口查询 Automation 服务器上允许访问的对象的描述。这种能力开创了一个使用 Access DBMS 进行数据库开发的全新的典范。

OLE DB 和 ActiveX 数据对象(Data Object),简称 ADO，试图取代数据访问对象(DAO)和远程数据对象(RDO)的 API。OLE DB 和 ADO 在数据提供程序与使用者之间提供了一个非常灵活的通用数据模型(COM)接口，使得 ADO 开发应用程序的速度更迅速。

本书针对以上四个方面，分别讲述了它们的基础知识和如何编写它们的数据库应用程序，并在书中讲述了大量数据库编程实例和技巧。

本书由孙景利策划，由杨浩广主编。另外，李锐、林乐、高翔、刘子锐、黄少棠、黄瀚华、凌贤伍、瞿小玉、孙宝玉、黄昌明、王洪秀、田尊五、吴广志、刘世德、李小峰、汪春军、张志明、王国戟、朱黎、陈果、李剑、董团结、顾云飞、刘贤铁、尹之恒、蒋伟峰等也参加了全书的编写工作。

编　者

2000 年 3 月

目　录

第一章　数据库基础

本章提要：

- 关系型数据库
- 数据库服务器
- 事务处理
- 结构化查询语言

数据作为表达信息的一种量化符号，正成为人们处理信息时的重要的操作对象。数据库根据数据和应用程序的相互依赖关系、数据共享以及数据的操作方式，对数据进行访问和管理。

数据库的应用极为广泛，几乎所有的部门都要进行数据库管理。特别是将网络技术和数据库相结合，更是提高了数据库的应用性。

Visual C++ 为数据库开发人员提供了方便的数据库开发接口。例如，Visual C++ 的 MFC(Microsoft Foundation Class)包含的建立在 ODBC(Open Database Connectivity)和 DAO(Data Access Objects)系统上的类，使得数据库的开发变得很轻松。

更为神奇的是，使用 AppWizard 可以创建简单的数据库应用程序，而不需要写一行C++代码。

为了使读者对 Visual C++ 的数据库开发的基本原理有一个全面的了解，本章将讲述一些数据库的基础概念，若你是一个专业的数据库开发人员，可以跳过这一章。

1.1　关系型数据库

关系型数据库是按照关系型数据库模型建立的数据库。在关系型数据库模型中，数据被细分为多个单独的列表或者表格(Table)。

通常用一个简单的二维表格来描述一个关系，该表格分为两个不同的部分，一是表头部分，它描述关系的名称，又称表名；二是表格内容，它描述关系中的具体值，表格中的每一列对应一个属性。

如图 1-1，这是一张名为 TermCodeMaster 的表，它有 4 个字段：sTermCode，lDueInDays，lDiscountInDays，和 dDiscountPctg，即 TermCodeMaster 表的 4 个属性。表中的每一行叫做一个记录(Record)，此时的表中有 4 个记录。

表与表通过关键字相关联，一张表中每个记录的唯一标识称为主关键字，并非所有的表都需要主关键字，但大多数表都具有。

一张表中的外来关键字是另一张表的主关键字。关系数据模型有助于减少重复数据并因此节省空间、时间和减少失败。

当增加与现有表相关联的其他表时，关系数据库的威力就显示出来了。因为现有表中记录的重复的数据可以包含在另一张表中。如学生的班级信息、所在城市等。这样会节省许多空间和消除许多录入时出现的错误。

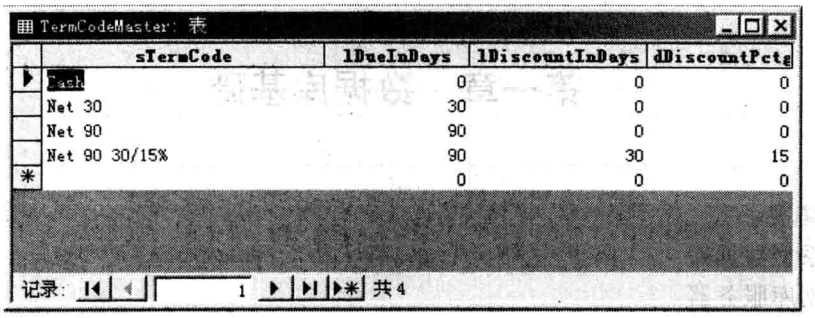


图 1-1 表格中的每一列对应一个属性

当需要根据一张表的信息去取出另一张或几张表中的信息时，要通过某种操作方式，这个操作称为联合(Join)。联合操作把至少有一个字段相同的表中的数据关联在一起——一个表中的外来关键字与另一张表中的主关键字相关联。

如图 1-2 所示，表 Permissions 通过外部关键字 sUserID，sCompanyID 和 sSubsystemID 分别与表 UserMaster，CompanyMaster 和 SubsystemMaster 连接(Join)。

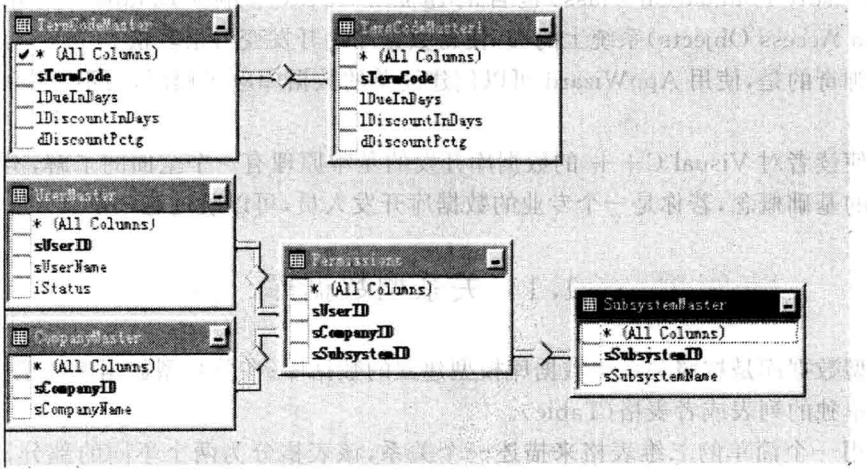


图 1-2 一个表中的外来关键字与另一张表中的主关键字相关联

1.2 数据库服务器

数据库服务器能容纳许多不同的表，但正常情况下它们不是以单个实体的形式存在的，而是被集中在一起放在两个或多个表的数据库或数据源中。一个包含公司运转过程的复杂数据的大型数据库，可能有几百个表来存储与该过程相关的所有类型的数据。

这些表及其中的组织叫做数据库的模式，它是由数据库设计者决定的。

一个数据库中，表的集合在数据库服务器看来是一个实体，因此服务器能为这个实体提供一些有用的服务。

例如，数据库服务器能维护任何数据库中的引用完整性。也就是说，数据库服务器将保证

外来关键字所列的每一项都在对应表的主关键字中有一个真正的同伴。

服务器还能使用事务(Transaction)提供多个表的数据库保护。例如,大型数据库中的单个复杂操作可能同时需要更新5个表。

但是,如果该操作开始后,只更新了两个表就因电源故障而无法继续,那么可以把对5个表的操作放在一个事务中,并通知数据库该事务开始执行5个操作,然后结束。如果在数据库服务器收到结束信号之前发生电源故障,服务器将把数据库恢复到该事务开始之前的那个状态。

数据库服务器可以控制多个用户或应用程序对数据的操作。

1.3 事务处理

事务处理是一个工作单元,它的成功与失败都必须作为一个整体来对待。例如,如果一位顾客把他的钱从一个银行账号转到另一个账号,那么这笔钱就必须从一个账号取出并存入另一个账号,决不允许这两个操作中有一个成功而另一个失败。否则,顾客的钱可能不公平地消失或不公平地得到一笔不义之财。

数据库编程人员很早就注意到了这个问题,并且开发了重新运行只有一部分成功的事务处理方法,即只有当所有的操作都成功时才开始事务处理。

大型的分布式系统中,事务处理就更为重要。例如,假定有两个准备从同一个顾客的银行账号上取钱(假定都要求取100美元)的系统,两个系统都通过网络连接到银行的保存收支平衡记录的系统中。第一个系统询问收支平衡时得到一个回答:账号上有150美元,稍后,第二个系统询问时也被告知账号上有150美元;第一个系统发出提取100美元的请求获得成功,而后的一瞬间第二个系统发出提取100美元的请求则失败。第二个系统在遇到失败以前已经完成的关于顾客的事务处理的任何部分都必须重新运行。

1.4 结构化查询语言(SQL)

以前,每种数据库系统都有它自己的编程接口,当你每次想查看以不同方式运行的数据库时,都必须进行相当复杂的数码之间的转换。为使这项工作简化,SQL,即结构化查询语言诞生了。

这种语言允许用户用简单的语句定义查询,以最适合需要的方式修改数据。虽然SQL是一种简单的语言,但它强大的功能足以使你完成对数据库操作的各种任务。

因此,它成为数据库编程、最终用户、查询等使用最广泛的语言,许多不同的数据库系统都支持SQL语言。

几乎所有的数据库都提供了它们自己的SQL语言,而这些SQL语言的使用略有不同的语法,支持略微不同的一套特征。

SQL语言能够在任何服务器上检索数据,向具体的表增加数据和更新表中的字段。SQL简单但功能强大。

SQL描述的三种基本语句形式:数据定义语言(DDL),用来建立数据库结构;数据控制语言(DCL),用来处理用户对某个对象的允许;最后,也是最重要的,数据处理语句(DML),用来

做其他任何一件事情，包括增加、修改数据和执行查询。这些语句是 SQL 的一部分，而且不是孤立的，虽然许多应用程序只用到了其中的一部分。

SQL 是一种访问数据库的方法，它可以交互式地、也可以通过应用程序来访问数据库，它被设计得读起来像英语。

大多数 SQL 语句都是查询语句，用于从一个或多个数据库中获得信息，但也可以使用 SQL 实现信息的添加、删除及修改操作。

下面讲述一些 Visual C++ 中常用的 SQL 语句。

1.4.1 SELECT 语句

SELECT 语句是 SQL 语句的灵魂所在，当你想从数据库中取出数据时，就会用到 SELECT语句。它的基本语法十分简单，虽然有许多附加的内容，这在以后可以看到。SELECT 语句最简单的形式为：

SELECT select-list FROM table-name

Select-list 可以有不同的形式，可以是最简单的，也可以是最通用的“*”。SELECT 将从选择表中选择所有的列。像下面的例子，它将对表中的每一行检索 MyTable 中的所有列：

SELECT * FROM MyTable

还可以从表中选择指定的一个或多个字段。例如：

SELECT EmpId, EmpName FROM MyTable

另外，可以在 seleclt-list 中定义固定的文本，这在 C++ 应用程序中非常需要，而且在报告生成效用方面（加入固定文本后的格式效果）是很常用的。例如，下面的语句将返回“----------”，仿佛它是另外一列的值。

SELECT EmpId , '----------',EmpName FROM MyTable

虽然在 C++ 应用程序中可以找到较好的格式数据，但这在向用户提供结果时是非常有用的。

1.4.2 WHERE 从句

在上面的例子中，我们检索了表中的所有行，你可以用 WHERE 从句限制这些行的返回。为了选择返回的行，WHERE 可以规定许多不同情况的表达式。最简单的表达式有下面的比较操作符：

小于 <

大于 >

不大于 <=

不小于 >=

等于 =

不等于 <>

例如，若 Employee ID 为 243，为此，用下述的 SQL 语句检索所有的行：

SELECT * FROM Employee WHERE EmpID =243

可以用 AND 或 OR 把几种情况合并起来，例如：

SELECT * FROM Employee WHERE Salary > 5000 AND Salary < 6000

SELECT * FROM Employee WHERE Dept='MIS'OR Dept='Sales'

此外，可以用 NOT 操作符否定某种状态，如下述例子中，会选出不在 Human Resources 的工作者的行：

SELECT * FROM Employee WHERE NOT Dept='HR'

这个简单的例子相当于下述的使用不等式操作符的语句：

SELECT * FROM Employee WHERE Dept<>'HR'

在单列名字的位置上，可以使用多列的表达式，多列用+，-，/，*连接。表达式可以用在 select-list 中，例如：

SELECT PartNum , PartCost + ExtraCost FROM Parts

WHERE 也可以包含表达式，例如：

SELECT PartNum FROM Parts WHERE (ExtraCost/PartCost)> .25

1.4.3　LIKE 谓词

除比较操作符以外，SQL 为字符串提供一种专用的比较操作符。LIKE 允许你选择与某种形式相匹配的字符串的行。在这种形式中，你可以包括任何正常的字符串，包括%和_。如果一个字符串中含有%和_，例如，选择 Title 中包含 Database 的行：

SELECT chapterNum FROM Chapters WHERE Title LIKE '%Database%'

也可以寻找在第二个字符位置上有 ata 开头的 Title，如：

SELECT ChapterNum FROM Chapters WHERE Title LIKE '_ata%'

1.4.4　IN 谓词

可用 IN 谓词简化一些 WHERE 从句，该从句用某一组值中的一个值来选择行。例如：

```
SELECT EmpNum FROM Employee
    WHERE Dept ='MIS'OR Dept ='HR'OR Dept ='Sales'
```

可以用谓词简化为：

```
SELECT EmpNum FROM Employee
    WHERE Dept IN ('MIS','HR','Sales')
```

你可以使用 IN 谓词的否定用法，下面所示的例子用来找出不在某一部分工作的雇员：

```
SELECT EmpNum
    FROM Employee
    WHERE Dept
    NOT IN ('MIS','HR','Sales')
```

1.4.5　BETWEEN 谓词

在许多情况下，需要选择列值在某范围的行，例如：

```
SELECT EmpNum FROM Employee
    WHERE Salary > 20000 AND Salary <30000
```

这个查询可用 BETWWEEN 简化，例如：

```
SELECT EmpNum FROM Employee
```

```
WHERE Salary BETWEEN 20000 AND 30000
```

像其他比较操作符一样，也可以把 BETWEEN 用在非数字列中，例如：

```
SELECT EmpNum FROM Employee
    WHERE Name BETWEEN 'Andersen' AND 'Baker'
```

为了选择没有在给定范围内的行，你可以加 NOT 调节器以拒绝 BETWEEN 谓词。

1.4.6 DISTINCT 关键字

在许多情况下，你想找到在一组行中获得的一个特殊列所需要的所有可能的值，可用 DISTINCT 关键字做到这点，它会删除结果中多余的列，如：

```
SELECT DISTINCT Dept FROM Employee
```

如果没有 DISTINCT 关键字，结果中的每行将包含所有列，而你关心的只是 Dept 列。

1.4.7 ORDER BY 从句

缺省时，通常查询返回的行是任意顺序的，如果你想让行按特定的顺序返回，需要用 ORDE1R BY 从句，它允许你指定一个或多个列用来整理返回的行。对每个列，可用 ASC 或 DESC 指定上升或下降顺序。

下面的例子将返回按 Employee ID 下降，Salary 上升顺序整理的行：

```
SELECT Empld FROM Employee ORDER BY Salary ASC, Empld DESC
```

1.4.8 集合

在许多情况下，你可能要在由结果集中返回的所有行的基础上计算数值，例如，在所有返回行中的一个列的总数。这些计算可以用集合函数实现，如下所示：

AVG：一个列值的平均数；

COUNT：返回行的数目；

MAX：在结果集中列的最大值；

MIN：在结果集中列的最小值；

SUM：在结果集中这些列值的总和。

也许最常用的集合是得到返回行的数目，下面的例子将返回 Employee 表示行的数目：

```
SELECT COUNT ( * ) FROM Employee
```

如果你用了 COUNT (*)，查询中所用的行都被计算。但是，如果指定了一个列的名字，只有列值不为空的行被计算。例如，在一个或多个行中 DEPT 列的值为空，那么下面的查询语句仅返回有值的 DEPT 行的数目：

```
SELECT COUNT (Dept) FROM Employee
```

在许多情况下，要在结果的某组行中使用集合功能，这可通过加入 GROUP BY 做到，它可以指定用来组成行的列，而这些行包括在集合计算中。下面的例子将产生一个 department 代码表，随后是每个 department 的 salaries 总数：

```
SELECT Dept , SUM (Salary) FROM Employee GROUP BY Dept
```

前面你看到了 WHERE 从句是如何限制查询的结果中返回的行；与之相似，加入一个 HAVING 从句，可以限制用集合功能返回的行。下面的例子将只返回一个总工资数少于

$200 000的 department 的工资总数：

```
SELECT Dept , SUM (Salary) FROM Employee
    GROUP BY Dept HAVING SUM (Salary) <200000
```

在更复杂的查询中，弄懂 WHERE 和 HAVING 从句如何一起使用是比较困难的，当对集合进行操作时，可按下面步骤进行：

(1) 选择符合 WHERE 从句的所有的行；

(2) 计算集合的值；

(3) 用 HAVING 从句限制了由集合得出的行。

1.4.9　ODBC SQL 文本值

在前面提到的例子中，你看到了在SQL 语句中使用简单的文本或常量值。对数字而言，不论是整型还是浮点型，可以用小数点区别。对于字符串，需用单引号把字符括起来。记住在SQL 语句中引用文本务必使用单引号而不是双引号；否则，编辑器会引起混乱。当把一个列设为 NULL 值时，可把 NULL 用作一个文本。

现在，有些文本感兴趣的地方是设置日期和时间。几乎每个数据库系统都有一套完全不同的方式来设置这些数据类型。ODBC 所用的 SQL 可提供一个方便的语法来指定这些值。

对于日期文本，使用下列形式：

```
{d 'yyyy- mm-dd'}
```

同样，对于时间文本，使用下列形式：

```
{t 'hh:mm: ss'}
```

用下列形式表示 timestamp 值：

```
{ts 'yyyy - mm - dd hh: mm: ss'}
```

1.4.10　联合

对关系数据库最重要的一种操作是联合，它返回的是从几个不同的表中集合起来的数据行，这些不同的表是在查询过程中相互联合的。如果你用的数据库是标准化的，那么联合更重要。

下面看一个例子，假如你有一个关于 parts 信息的 parts 表，包含 descriptions，还有一个关于 parts 价格信息的 Prices 表，下面的查询将对两个表进行联合操作：

```
SELECT Description , Price FROM Parts, Prices
```

然而，你可能对结果感到惊奇，上面的例子返回两个表的 Cartesion 结果，这个结果是 Parts 表和 Price 表的所有行可能结合的名字，在很多情况下，这个结果无用。

为使结果有用，可以加入 WHERE 从句。最常见的情况是两个表中有共同的列。例如，假设 Price 表和 Part 表都包括 PartNum 列，可以用下面的查询产生 parts 的despription 和 price 匹配的行：

```
SELECT Description , Price FROM parts , Prices
    WHERE Parts. PartNum = Prices. PartNum
```

1.4.11 相关的名称

为了简化一些语句,SQL 允许使用相关的名称,可以省略长的语句,下面你会看到这对于复杂的语句是必要的。如:

```
SELECT MyFirstTable. Name, MyFirstTable. Num,
    MySecondTable. Date,MySecondTable. Time
    FROM MyFirstTable, MySecondTable
    WHERE MyFirstTable. ld = MySecondTable. ld
```

我们使用相关的名称,用 f 代替 MyFirstTable,用 s 代替 MySecondTable:

```
SELECT f. Name, f. Num, s. Date, s. Time
    FROM MyFirstTable f, MySecondTsble s,
    WHERE f. ld = s. ld
```

这时,相关名称不影响语句的意思。在更复杂的语句中,你可能用了很多次同一个表,这时,相关名称可简化你用到的表。

1.4.12 外部联合

到目前为止,你看到的仅是内部联合,它可能不会产生联合表的 Cartesian 结果。如果用下面的语句会产生一个联合:

```
SELECT * FROM Employee, Department
    WHERE Employee. DeptNum = Department. DeptNum
```

数据库会收集 Emploree 表中所有的行与 Department 表中的行相匹配。因为没有对应的入口在 Employee 表中,所以不会有结果。为了产生这样的一个查询结果:返回的行不包括在一个表中匹配的行,应使用外部联合。

例如,设想你要为每个 employee 选择行,包括没有分到一个 department 中去的,可以这样使用外部查询:

```
SELECT * FROM
    {oj Employee LEFT OUTER JOIN Department ON
        Employee. DeptNum = Department. DeptNum}
```

在这个例子中,我们用到了 LEFT OUTER JOIN 以确保左边表 (Employee) 中的所有行都在结果中。

还可用 RIGHT OUTER JOIN 以确保右边的表,或用 FULL OUTER JOIN 以确保两个表中的所有行都在结果中,不管是否有匹配的行出现在另外一个表中。

1.4.13 子查询

在我们看到的 WHERE 从句的例子中,已比较了文本值或比较了两个列。其实,还可以在 WHERE 从句中用另外一种查询得到信息,这就是子查询。使用子查询的第一种情况是用 EXIST (或 NOT EXIST)谓词。设想挑选一些没有 employee 的 department:

```
SELECT DeptName FROM Department
    WHERE DeptNum NOT EXIST (SELECT * FROM Employee)
```

这将返回所有 department 的名字，因为在 Employee 表的记录中没有找到 department number。

还可以挑选 departments 的表至少有一个高工薪的 employee 的 department，这个查询如下：

```
SELECT DeptName FROM Department d
    WHERE EXISTS
            (SELECT * FROM Employee e
            WHERE e.Salary > 100000 AND d.DeptNum = e.DeptNum)
```

使用子查询的第二种情况是用 IN 或 NOT IN 谓词，例如，可以选择一些目前有 employee 分配去的 department：

```
SELECT DeptName FROM Department
    WHERE DeptNum IN (SELECT DeptNum FROM Employee)
```

第三种使用子查询的情况是使用 ANY 或 ALL 关键字的比较。例如，有独立的 employee 表和 executives 表，想任选一些比普通 employee 报酬高的 executives，可用下面的查询：

```
SELECT Ex.Name FROM Executives EX
        WHERE Ex. Salary > ALL
                (SELECT Emp.Salary FROM Employee Emp)
```

同样，也可用 ANY 修饰语选择一些 executives，至少有一个普通 employee 的报酬比他高：查询如下：

```
SELECT Ex. Name FROM Executives Ex
    WHERE Ex. Salary < ANY
            (SELECT Emp. Salary FROM Employee Emp)
```

1.4.14 联合查询

SQL 允许从两个独立的查询中通过 UNION 关键字得出一个结果。缺省时，双重的行可从结果中移走，虽然可用 UNION ALL 而不是 UNION 来终止。

还有，你可以指定一个 ORDER BY 从句来选择最终的结果，在联合中，在最后查询中使用 ORDER BY 从句，例如，可用下面的查询得到一个在 MIS 和 Salary department 中 employee的记录：

```
SELECT * FROM Employee
WHERE Dept = 'MIS'
UNION
    SELECT * FROM Employee
    WHERE Dept ='SALES'
```

当然，可以有几种方法产生查询的结果，这是一个 UNION 使用的简单的例子。

1.4.15 INSERT 语句

你已看到了如何用各种不同的方式从数据库中得到行，那么如何以第一位的方式把行放进表中呢？这要用到 INSERT 语句。INSERT 语句的基本语法是：

```
INSERT INTO table — name [ (column — list) ]
    VALUES (value — list)
```

如果要为每个列加入值，遗漏了列表和对每一列的简单赋值，那就要按照列在表中出现的顺序来进行，如下例：

```
INSERT INFO Employee
    VALUES (123, 'Bob Jones', 35000, 'MIS')
```

如果要为列赋值，应列出这些列，所赋的值必须与列对应。当表中某个列为缺省值或改变某个列的 NULL 值时，这非常有用。下述的例子将加入一个新的 employee 记录，舍去 Salary Null：

```
INSERT INTO Employee (Empld, EmpName, Dept)
    VALUES (123, 'Joe Bob Griffin', 'Sales')
```

此外，如果不用 VALUES，从句可以在 SELECT 语句返回值的基础上加入行。例如，你的公司扩展了 MIS 部门，你可以在 FoemerEmployee 表中用下面的语句加入新的记录：

```
INSERT I NTO FormerEmployee (Empld, Empname, Reason)
  SELECT Empld, Empname, 'Outsourced' FROM Employees
    WHERE Dept = 'MIS'
```

1.4.16 DELETE 语句

你已加入多个行在表中，如何删除它?DELETE 语句可做到这点。基本的 DELETE 语句非常简单，可以删除整个行，因此没有必要指定单个的列，基本语法为：

```
DELETE FROM Employee
```

但是，这个简单语句非常强大，它删除表中所有的行，在很多情况下，可用 WHERE 从句删除指定的行：

```
DELETE FROM Employee WHERE Empld = 456
```

1.4.17 UPDATE 语句

UPDATE 语句允许你修改数据库中现存行的值，如：

```
UPDATE Employee SET Salary = Salary + 100
```

可用 WHERE 从句限制受影响的行，如果用下面的语句限制最小 Salary 数：

```
UPDATE Employee SET Salary = 21000 WHERE Salary < 21000
```

当然，也可用 UPDATE 一次修改多个列，或使某个列为 NULL。在下面的例子中，Mary 结婚了，改变了她的名字，出外度蜜月。我们可以改变 name ，把她的 department 设为 NULL，改变她目前的 status：

```
UPDATE Employee
  SET Employee = 'Mary Jones', Dept = NULL, Status = 'On Leave'
  WHERE EmpNum = 324
```

1.4.18 调用过程

许多数据库系统允许你用 SQL 调用，由系统自己定义的过程，通过使用存在服务器上的

过程，会使你的应用程序更有效，而不用担心陷入网络阻塞。

为了调用过程，使用下面任何一种语句：

{call myProcedure}

在许多情况下，调用过程会返回一个值或需要一个或多个输入输出参数，我们将在后面的章节中看到每种数据库接口如何处理参数，而且允许你用“?”标出参数的出入位置，如：

{call ? = PromoteEmployee (?,'Manger', ?)}

上面的例子给出了3个参数：一个返回参数，两个输入参数。