第 7 章
了解数据库

关系型数据库(relational database)的中心思想是以关系型模型的概念,套用至原本看来彼此无关的数据,也就是为数据加上关联式模型的定义,这个定义包含分隔的数据实体(data entities)、实体属性(attributes)、实体间关联(relationships)。

7.1 了解关系数据库管理系统

数据库管理系统(DBMS)是为管理数据库而设计的电脑软件系统,一般具有存储、读取、安全保障、备份等基础功能。常见的 DBMS 有 MySQL、Microsoft Access、SQL Server、Oracle 等。几乎所有的数据库管理系统都配备了一个开放式数据库连接(ODBC)驱动程序,令各个数据库之间得以互相整合。

7.1.1 数据库产品的特性和功能

数据库是数据的集中存储位置,且数据库所含数据之间必定具有关联,借助于关联的定义,将多笔、多样的不同数据集中在数据库中,成为一个紧密结合的处理单元。

使用数据库的目的是可以集中存储、管理使用单位的重要资料,若一个数据库经过良好的设计,应该可以提供多个使用者,在多个应用系统中执行存取的处理动作。为了完善数据库应用,通常会使用数据库管理系统(DBMS),针对数据库执行存储、存取、保护、安全等作业,而数据库管理系统的设计与操作,可以使用目前常见的数据库软件,包括 Microsoft SQL Server、Microsoft Access、Oracle、MySQL 等。

数据库管理系统可以根据不同的数据模型而完成,但数据库应用发展至今,关系型数据模型几乎已是市场标准,所有可见的数据库软件,都是以关系型数据模型为基础,这些以关系型数据模型为基础的数据库管理系统,可称为关系型数据库管理系统(Relational DBMS,RDBMS),常见的 SQL Server、Access、Oracle、MySQL 等都是 RDBMS。在这个模式中,数据是存储在数据表(Table)内,每一个数据表又可以包含多笔记录,或称为数据行(Row)。

还有一些以其他模型为基础的数据库管理系统,如面向对象数据库管理系统(Object DBMS,ODBMS),基于面向对象模型数据的存储方式采用面向对象的原理,成为一个个

东软电子出版社

不同的对象。

关联式 DBMS 的特色是使用结构式查询语言(Structured Query Language,SQL),执行对于记录的各项处理,目前大部分关联式 DBMS 都支持标准的 SQL 语法,所以学会 SQL 语法,就可以使用多种不同关联式 DBMS 处理及管理数据库。

关联式数据库对于数据的存储及呈现是采用二维形式,也就是数据表、记录及字段等三种组成。关联式数据库将数据存储在数据表,一个数据表的结构是由记录及字段所组成,类似于 Excel 的工作表。字段本身定义了数据类型,表示在该字段中的数据的可以接收的数据值的范围,例如客户的订单可以保存名为 Orders 的数据表,每条记录都代表一笔订单,其内的 OrderData 字段记录每笔订单的订单日期,如图 7-1 所示。

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
▶	10248	VINET	5	1996-07-01 00:00:00.000	1996-08-01 00:00:00.000
	10249	TOMSP	6	1996-07-05 00:00:00.000	1996-08-16 00:00:00.000
	10250	HANAR	4	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000
	10251	VICTE	3	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000
	10252	SUPRD	4	1996-07-09 00:00:00.000	1996-08-06 00:00:00.000
	10253	HANAR	3	1996-07-10 00:00:00.000	1996-07-24 00:00:00.000
	10254	CHOPS	5	1996-07-11 00:00:00.000	1996-08-08 00:00:00.000

1 /7

图 7-1　数据表范例

7.1.2　数据库设计

一般而言,关系型数据库的使用对象是商用系统,设计过程必须遵循一定的程序,以确保设计完成的结果,可以满足所需。

首先必须了解的是数据往往是使用单位的重要资产,这项资产必须有再利用的价值,这也是使用关系型数据库的目的,在定义数据库相关规格时,第一重点是资料的整合性,也就是由微观角度而言,个别数据在数据库内虽然是分别存储,但站在宏观角度时,又希望这些个别资料之间的横向关联必须正确或者必须在特定时机有所关联。以下是设计数据库的一般步骤:

(1)列出或写下数据库的任务,以叙述方式予以条列清楚,这个工作可以协助分辨及使用数据库的目的,包括谁来使用、如何使用等,同时此时定义的内容也会成为日后发展数据库应用的大方向。

(2)在此阶段确定哪些数据需要存储?哪些不需要存储?需要存储的资料中,又应使用什么数据类型?一连串的问题由此开始。另本阶段的分析,也可以使用需求分析的结果,如实体关系图等作为辅助。

(3)将属性相近的数据予以集中,成为一个数据表,再在数据表中划分多个字段并为每个字段定义数据类型。

(4)在多个字段中选择一个主索引,可以是一个或多个字段组成主索引。主索引的值在数据表内必须唯一,不可重复,目的是可提供系统对每笔不同记录的识别。

(5)在多个数据表之间,找出应该建立的关联,并赋予实际意义。关联双方必定是两边数据表的一个字段,两个字段的数据会因为关联而产生意义,如 Customers 资料表存储

东软电子出版社

多笔记录，代表不同客户，Orders 存储的多笔记录代表来自客户的订单，所以这两个数据表必须是一对多关联，一是 Customers，多是 Orders，因为一个客户可以多次下单。

(6)将前步骤的结果，根据正规化的要求，予以正规化，目的是将分析后的结果，予以明确划分及定义，符合关系数据库的原理，包括数据正确性及完整性。

以上共六个步骤，步骤 1 可以视为准备工作，步骤 2 至步骤 5 可以说是关于关系型数据库的系统分析工作，而步骤 6 等于产生分析结果，作为下一步设计的参考。

7.1.3 实体关系图(ERD)

实体关系图是以图形的方式，呈现关系式设计的结果。主角是实体(entity)及关系(relationship)，多个实体间会因不同行为或状态而产生关系，借助实体关系图的图形化呈现，可以一目了然地检验结果，也可由此确认数据本身的内容，是否需要存储在数据库。

在分析阶段，通常必须借助图形化的辅助工具协助分析，实体关系图就是在数据库的前置作业期间，经常使用的图形。它使用多种不同符号，代表各自意义，展现实体、属性及关系等三者之间的对应，以下是这三个组成的说明：

(1)实体(Entity)：一个实体代表的是真实存在、不可见的概念或可见的实体，例如一笔订单、一个客户、一位员工等，这些个别存在的资料都是一个实体。

(2)属性(Attribute)：这是对于实体的描述，形容实体如何存在于数据库内，对于订单而言，可能的属性有订单编号、订单日期、运送日期、销售金额、出货方式等；对于员工而言，可能的属性有员工编号、姓名、职称、到职日期等。且每个实体都必须有一个关键属性，可以用做唯一标识之用，称为主索引(primary key)，例如订单实体的订单编号就可以是主索引，因为不同订单的编码必定不同，故经由订单编码找到的订单必定只有一笔。

(3)关系(Relationship)：也可称为关联，它是在两个实体之间，予以连结的设计，例如稍早说明的订单与客户，再如订单与员工之间，也应该有关联，因为必须记录哪一位员工处理订单。

实体关系图展现的是上述三个组成的定义，而非数据本身。如有 1000 笔订单及 300 笔客户，代表有 1000 个订单实体及 300 个客户实体，但在实体关系图中，只会显示客户、订单实体及其关联的结构。而 1000 个订单及 300 个客户实体的实际关系，如“客户 A 在今天的三笔订单”，这是查询，而非关系，实体关系图展现的是“客户与订单的关系要如何定义”。

以下是使用于实体关系图的常见符号：

- 矩形代表实体
- 椭圆代表属性
- 菱形代表关系
- 以直线连接上述三个组成，如实体及属性，实体与关系

如图 7-2 所示，这是一个实体关系图的范例，此图表示共有两个实体，分别是 Customer 及 Order，且各有三个属性，每个属性各自连至所属实体。Customer 实体的属性有 ID、Name、City 等，Order 实体的属性有 OrderID、OrderDate、ShipDate 等，多个属性中，会有一个成为键属性(key attribute)，键属性的目的是用来识别唯一实体，所以 ID 及

东软电子出版社

OrderID 是两个实体的键属性。另还有一个在 Customer 及 Order 之间，名为 Place 的关系。

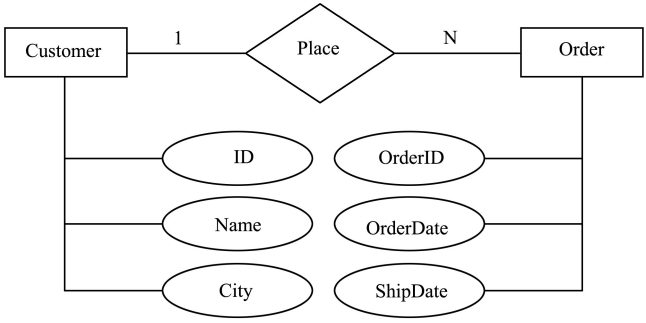

图 7-2　实体关系图范例

关系依据不同特性又分为一对一、一对多及多对多等三种类型。图 7-2 关联线在 Customer 端，显示“1”，在 Order 端则为“N”，表示在 Customer 及 Order 实体间的 Place 关系，是一对多类型。以实例而言，表示一个客户可以下单多次，所以由一个 Customer 实体角度来说，可以对应到一个或多个 Order 实体，以一个 Order 实体来说，就必定只会对应到一个 Customer 实体。在实例设计中，绝大多数的关系都是一对多，一对一及多对多都很少使用。

以下是将实体关系图转换为关系型数据库的步骤：

(1)处理实体：为每一个实体建立个别数据表，属性成为字段，键属性成为主索引，若需要使用多个属性的组合，成为键属性，就同时将各个属性对应的字段组合成为主索引。

(2)处理关系：关系在落实至数据表时，关系的双方必须是两个数据表的各一个字段。若是一对多的类型，此时在一方必须使用该资料表的主索引字段，在多方必须不是主索引字段。如图 7-2 所示，必须先在 Order 数据表增加一个字段，假设名为 CustomerID，这个字段就是在关联中作为多方代表的字段，而一方的字段，必须是 Customer 数据表的主索引，如此即可完成一对多关联。此时位于 Order 数据表的 CustomerID 字段可以称为外部索引键(Foreigh Key)，因为它的资料是外来的，是来自关联的另一端。完成一对多关联的设计后，就可以由客户取得订单，或反过来在订单中获得客户数据。

所以如图 7-2 所示的实体关系图，在转换至关联式数据库后，会产生表 7-1 所示数据表及表 7-2 所示关联。

表 7-1　　**Customers**

ID	Name	City
1001	Jane Doe	Berlin
1002	John Doe	Tokyo
1003	Howard Steel	Sydney

东软电子出版社

表 7-2 Orders

Order ID	CustomerID	OrderDate	ShipDate
101	1001	10/1/2010	10/7/2010
102	1001	10/5/2010	10/10/2010
103	1001	10/4/2010	10/10/2010

7.1.4 了解规范化

规范化是针对前一个阶段的分析结果,再进一步的处理,目的是检查分析结果中有没有可能导致数据不一致、违反完整性的情况,若有则予以更正。

实体关系图的分析可以产生关联式数据库所需的结构,但产生的结构不一定正确。因为实体关系图的重点不在于实体本身,所以必须以规范化的原则检查实体关系图,确保最终的数据库结构是符合关联式数据库的处理原则,避免未来因结构不正确,而可能造成的问题。

以表 7-3 的数据表为例,这是 Books 实体转换而来的资料表,为了说明正规化步骤,我们先在这个资料表加入数条记录。

表 7-3 Books

Book ID	BookName	CategoryID	CategoryName
1	Cooking Ligh	1001	Cooking
2	Prophecy	1002	Mystery & Thriller
3	Shift	1003	Business
4	The Confession	1002	Mystery & Thriller

在表 7-3 的结构及资料中,会导致如下三个问题:

(1)新增异常:表示在新增记录时,会因为不明确的数据依据,产生违反数据完整性的异常,如若要在上表加入历史类别的新书籍,就无法操作,因为目前的书籍类别中没有历史一项,因为目前的 CategoryID 及 CategoryName 等字段是在 Books 数据表内,无法加入不存在的新类别。

(2)删除异常:表示在删除记录时,会因为不明确的数据依赖,发生违反数据完整性的异常,如若要在上表删除 BookID 字段等三条记录,则 Business 类别的数据也会丢失。

(3)更新异常:表示在更新记录时,会因为不明确的数据依赖,发生违反数据完整性的异常,如若要在上表中,将所有 Mystery 及 Thriller 类别的名称改为 Mystery,就必须在多笔记录执行更改,而此一更改就有可能因为输入错误,导致数据不一致,也就是更新动作以精确、少量为原则。

规范化的目的就是解决上述三个可能的问题,在理论上,规范化共有五个层次,由第一个阶段到第五个阶段,但常用者只有第一至第三阶段。

规范化的动作是由数据分析开始,可以协助开发人员确保数据库结构的正确性,请注意是针对结构而非数据本身的内容。

1. 第一阶段规范化

这是最基本的规范化原则,第一阶段正规化(firstnormal form,1NF)的原则是字段内

东软电子出版社

只能含有单一值，如在以下的 Customer 数据表中，PhoneNumber 字段的第二及第三笔记录，都各存储两个电话号码，这就是非单一值的数据，表示此字段的存储方式违反第一阶段规范化

表 7-4　　Customer

ID	FirstName	LastName	PhoneNumber
1	Jane	Doe	(010)234-5678
2	John	Doe	(011)345-3456, (010)567-3456
3	Howard	Steel	(010)678-4567, (010)456-4567

为了符合第一阶段规范化的原则，表 7-4 必须予以拆分，成为如下两个数据表：

表 7-5　　Customer

ID	FirstName	LastName
1	Jane	Doe
2	John	Doe
3	Howard	Steel

表 7-6　　CustomerPhones

ID	PhoneNumber
1	(010)234-5678
2	(011)345-3456
2	(010)567-3456
3	(010)678-4567
3	(010)456-4567

表 7-5、表 7-6 两个数据表分别是 Customers 及 CustomerPhones，且都含有各自的主索引字段，在 Customers 是 ID，在 CustomerPhones 是 ID 及 PhoneNumber，再以双方的 ID 字段建立关联，故在 Customers 数据表可以由关联找到每一客户存储在 CustomerPhones 数据表的电话号码，因为每个字段的数据都是单一值，如此即可以完成第一阶段正规化的检查和处理。

在以上的范例中，若在 Customers 数据表建立 PhoneNumber1 及 PhoneNumber2 等两个字段分别存储电话号码，虽然可以达到存储单一值的目的，但第一阶段规范化仍不允许这种浪费存储空间的设计。

2. 第二阶段规范化

进入第二阶段规范化之前，首先必须完成第一阶段规范化，或者以第一阶段规范化的结果，进入第二阶段规范化。第二阶段规范化的原则是所有非主索引字段的值，只能与主索引字段内容全功能相依。

首先要说明所谓功能相依，以稍早在第一阶段正规化的两个数据表为例，在 Customers 数据表中，ID 是主索引字段，它可以用来识别每笔不同的记录，所以有 ID 字段可以取得各个记录的 FirstName 及 LastName 字段值，且取出的数据必定因为 ID 字段

东软电子出版社

值的不同而不同，因为这两个字段都不是主索引字段之一，所以我们可以说 Customers 及 CustomerPhones 数据表都符合第二阶段规范化的原则。

第二阶段规范化通常会应用于以多个字段作为主索引时，此时是一个主索引，但每一主索引的值是由多个字段所组成，故组合后的值必须唯一，此时较容易发生违反第二阶段规范化的情况。反过来说，若只有一个字段作为主索引，且已符合第一阶段规范化时，也必定符合第二阶段规范化。

以下说明如何以一个有问题的例子，更正违反第二阶段规范化的错误，假设有一个 Orders 数据表见表 7-7。

表 7-7 CustomerPhones

Order ID	CustomerID	OrderDate	CustomerName
101	1	10/1/2010	Jane Doe
102	2	10/5/2010	John Doe
103	1	10/4/2010	Jane Doe

表 7-7 中，Orders 数据表的主索引是 OrderID 及 CustomerID，这两个字段值的组合必须唯一。所以在规划中，OrderDate 字段作为订单日期，其值会相依于 OrderID，因为订单日期必定会随着订单编号而不同，而 CustomerName 字段是客户名称，其值会相依于 CustomerID 字段，因为不同客户必有各自的名称，所以结果是 OrderDate 及 CustomerName 各自相依于不同的主索引字段，这就违反了所有非主索引字段之值，只能与主索引字段内容全功能相依的原则。全功能相依之意是所有非主键字段必须以同一个规则，相依于主索引。

对于表 7-7 的 Orders 资料表，更正的做法是移除 CustomerName 字段，使 Orders 数据表只有三个字段，就可以保证剩下的非主索引字段，即 OrderDate 只相依于 OrderID。修改的方法不是移除字段，就是更改结构。至于若是移除，应如何完成原来的目的，就不是第二阶段规范化的议题，视实际情况而定。

3. 第三阶段规范化

符合第二阶段规范化的数据表，可以再进行第三阶段规范化的检查。这个阶段表示不可有转接相依，也就是所有非主索引的字段不可相依于其他非主键字段，只能相依于主索引字段，如以表 7-8 的 Items 数据表为例：

表 7-8 Customer

ItemID	SupplierID	ReorderFax
101	100	(010)234-5678
102	11	(011)345-3456
103	525	(010)678-4567

其中，ItemID 是主索引字段，ReorderFax 是供应商的传真号码，且相依于 SupplierID。但这个字段不是主索引，故可以说 Items 数据表违反了第三阶段规范化。更正的做法是拆分为两个数据表，分别是 Items（ItemID，SupplierID）及 Supplier（SupplierID，ReorderFax），见表 7-9、表 7-10：

东软电子出版社

表 7-9　　Items

ItemID	SupplierID
101	100
102	11
103	525

表 7-10

SupplierID	ReorderFax
100	(010)234-5678
11	(011)345-3456
525	(010)678-4567

7.2　了解数据库查询方法

SQL 如今已成为关联式数据库管理系统的标准，它的主要功能是取得及更改数据，也可以执行建立数据表及针对数据库执行维护操作。在不同的关联式数据库系统中，它都可以看成是数据库的指令式工具，熟悉它，就可以通过指令操作及管理数据库。

使用 SQL 的处理，等于告诉数据库“请依据命令执行以下操作”，数据库引擎就会分析传入的 SQL 语法，只要语法正确，就予以执行。例如现要由指定数据表取得前十笔记录，若要使用 C＃语言，可能需要循环遍历所有的数据，就如同在一个阵列内，执行排序，而排序的每个动作又有切割、搬移、重置等，一系列动作都需要程序代码的处理。SQL 就只要一行即可，当然也有其限制，就是 SQL 的处理对象只可以是数据库。

SQL 最早是由美国国家标准协会(ANSI)所制订，又经过数个版本的改良，如今众多关系型数据库管理系统都已采用它作为针对数据库的标准处理语言。在 SQL Server 中提供的 SQL 语法称为 Transact-SQL(T-SQL)。

在 SQL Server 中，可以直接输入及执行 SQL 指令，也可以建立存储过程，在存储过程内书写 SQL 指令，此时可以集合多个 SQL 指令，在有需要时加以执行。

7.2.1　结构化查询语言(SQL)

SQL 语法中有关查询的指令主要有 SELECT、INSERT、UPDATE、DELETE 等，这四个指令的使用机会相当频繁，又以 SELECT 最常使用。

学习 SQL 语法的最简单方式是由查询操作开始，SQL Server 当然也提供了自动产生 SQL 语法的工具。本节中将学习以下四种 SQL 指令：

SELECT：可以由数据表取出记录，也就是查询。

INSERT：可以新增记录至数据表。

UPDATE：可以在数据表执行更新记录。

DELETE：可以在数据表执行删除记录。

1. 连接至数据库

若要操作以下示例，必须先连接至 SQL Server 数据库，作为操作处理的对象。操作

东软电子出版社

范例前，请先启动 Visual Studio。

(1)由 Visual Studio 窗口中选择“视图”菜单中的“服务器资源管理器”，选择“数据连接”节点，再选择“连接至数据库”按钮。

(2)在“数据源”对话框中指定数据源名称，在打开数据库名称清单，选择需要连接数据库名称，如图 7-3 所示。

图 7-3 连接至数据库

(3)请选择“使用 Windows 验证”，再按下“测试连接”按钮，确认可连接至 SQL Server 数据库，如图 7-4 所示，最后按下“确定”按钮。

(4)此时 Visual Studio 会由指定的数据库取得数据库的结构，并在服务器资源管理器中显示为多层清单的形式，其中较重要的节点是“表”及“存储过程”，如图 7-5 所示。

图 7-4 连接至数据库

图 7-5 连接至数据库后的服务器资源管理器

东软电子出版社

(5)选择服务器资源管理器中的数据库名称,在按下 F4 键开启属性表,可以看到“连接字符串”的值,这个值就是连接至 SQL Server 的语法,如图 7-6 所示。

图 7-6　数据库的属性表

2. 在 Visual Studio 中执行查询

可以使用相关工具提供的多种不同方法,针对 SQL Server 数据库执行查询。以下是一般设计中,针对 SQL Server 数据库执行查询的常用方法:

(1)Visual Studion 整合设计环境(Integrated Development Environment,IDE)

(2)C#应用程序

(3)SQL Query Analyzer

以下操作将在 Visual Studio IDE 中开发的 C#应用程序对数据库执行查询。

(1)在服务器资源管理器的数据库节点按下鼠标右键,选择菜单中的“新增查询”,可打开查询工具,并显示加入数据表的对话窗口,请选择数据表的名称,分别选择“加入”及“关闭”按钮。

(2)查询设计工具会显示由上至下四个区域,请在由上至下第三个区域中显示的 SQL 语句修改为:

```
SELECT * FROM Customers
```

(3)由“查询设计器”菜单选择“执行 SQL”,就会以设定的 SQL 执行查询,并显示结果,如图 7-7 所示。

如图 7-7 所示,Visual Studio 的查询设计工具会显示四个区域,以下是这四个区域的功能说明:

• 数据源表区域:此处会以图形显示查询使用的数据表及字段,若使用多个数据表,则资料表之间必须建立联系。

• 字段及条件区域:此处可以设计目前在查询结果中的字段,以及查询条件、排序设定等。

• SQL 语法区域:此处会显示以上两个区域的设定,并转换为 SQL 语法,若用户对 SQL 语法熟悉也可以在此输入 SQL 语法。

• 查询结果区域:若 SQL 语法的设定可以回传结果,会显示在此区域。

3. 在 C#应用程序中执行查询

以下操作说明如何在 C#应用程序中,针对数据库执行查询。

东软电子出版社

图 7-7 Visual Studio 的查询设计工具

(1)建立新的 Windows Forms 应用程序项目,命名为 QueryCS。

(2)在立即显示的 Form1 表单中,添加一个 TextBox、一个 Button、一个 DataGridVies 控件,再将 TextBox 控件的 MultiLine 属性设定为 True,表示允许输入多行文字。最后将 Button 控件的 Text 属性设定为 Execute SQL。

(3)在 Button 控件双击鼠标左键,进入代码编辑窗口的 Button 控件的 Click 事件中,再输入如下代码:

```
private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.TextLength > 0)
            {
                SelectData(textBox1.Text);
            }
        }
```

(4)在程序代码窗口加入名为 SeleData 的方法,并将其中的 selectConnection 的内容更改为目前的数据库连接字符串,即如图 7-6 所示的“连接字符串”。

```
private void SelectData(string selectCommandText)
    {
        try
            {
                string selectConnection = "Data Source = BTIC - TJ - 065;" +
                  "Initial Catalog = DataBaseTest;Integrated Security = True";
                SqlDataAdapter dataAdapter = new SqlDataAdapter(selectCommandText,
                  selectConnection);
                DataTable table = new DataTable();
                dataAdapter.Fill(table);
                dataGridView1.DataSource = table;
```

东软电子出版社

```
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

(5)在程序代码的最上方使用 using,加入如下两个引用设定:

```
using System.Data;
using System.Data.SqlClient;
```

执行项目,在 TextBox 中输入 SQL 语法,再按下 Execute SQL 按钮,就可以显示执行结果,如图 7-8 所示。

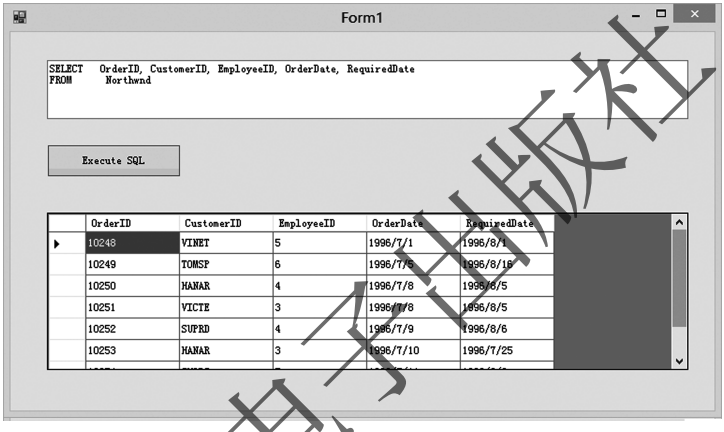

图 7-8　在 C# 应用程序中执行查询

以上范例的设计是使用 SelectData 方法执行查询,此方法内的处理流程是先使用 SqlDataAdapter 对象取得查询结果,再将结果置于 DataTable 对象,最后将 DataTable 的内容显示在 DataGrid 控件中。所以面对数据库及执行查询的是 SqlDataAdapter,它会以 Fill 方法将取得的内容放入 DataTable 内,而 SQL 语法则是置于传入 SelectData 方法的参数 selectCommandText 内。

4. 取得 SQL 数据

取得数据的命令是 SELECT,这也是最常用的 SQL 指令,以下是使用 SELECT 指令的常见用法:

```
SELECT   list_of_fields
FROM list_of_tables
WHERE where_clause
Group BY group_by_clause
HAVING having_clause
ORDER BY order_by_clause
```

以上是一个完整的 SQL 语法,以 SELECT 为首,每一行都可以视为一个子句(clause)。其中 SELECT 及 FROM 是必要子句,SELECT 之后是字段名称,FROM 之后是数据表名称,表示由(FROM)数据表取得(SELECT)一个或多个字段,如以下是最简单而完整的 SQL 语法:

东软电子出版社

```
SELECT   OrderID, CustomerID  FROM       Northwnd
```

若要由数据表取得所有字段，也可使用字符“*”，如：

```
SELECT   *   FROM     Northwnd
```

若要使用多个数据表执行查询，可将多个数据表名称写在 FROM 之后，如：

```
SELECT OrderId , Customers.CustomerId , ContactName
FROM Orders , Customers
```

以上的写法又稍有变化，因为在 SELECT 之后的字段名称，都加上了数据表的名称，成为如 Customers. CustomerId 的模式，这是因为在使用多个数据表作为数据源时，若多个数据表有同名字段，就必须予以明确指定，如 Customers 及 Orders. CustomerID。总之在使用多个数据表时，为所有字段指明数据表名称，可以避免可能的重名问题。但以上的写法其实也是不正确的，执行结果会比预期增加许多。这是因为两个数据表的所有输出字段，都会彼此结合(cross join)，结合的原因为虽然明确了两个数据表，但没有定义两个数据表之间的关联，所以在不知道如何关联的情况下，系统会为所有字段定义关联，也就是 join。以 Customer 及 Orders 为例，分别代表客户及订单，在正确关联时，用户会希望取得一个客户对应的订单，此时可使用 INNER JOIN 完成这个操作，如：

```
SELECT OrderId , Customers.CustomerId , ContactName
FROM Orders  INNER JOIN Customers
ON Orders.CustomerId = Customers.CustomerId
```

以上语法的重点是 FROM…INNER JOIN…ON…，表示要求 SQL Server 由两个数据表取得数据，且取出原则是由 Order. CustormerId 字段值必须等于 Customers. CustomerId，这两个字段其实就是两个数据表的关联字段。由于 Customers. CustomerId 是主索引，其值在 Customer 数据表中必须唯一，所以在查询结果中，可以取出每位不同客户的所有订单。在查询结果的 Customers. CustomerId 字段值可能会重复，但 OrderId 则不会重复，因为 OrderId 代表不同的订单号码。

一个查询语句多数情况下需要加上查询条件，查询语句只取出符合条件的记录，此时可以使用 WHERE 子句，条件的设定对象是字段，可以在特定的字段内查找符合条件的数据，如：

```
SELECT   *   FROM     Northwnd
WHERE ShipCountry = "Canada"
```

以上的例子中，系统会在 Orders 数据表每笔记录的 ShipCountry 字段值逐一对比，若数据内容等于 Canadan，就会显示查询结果，反之则不显示。若条件不止一个，也可以在 WHERE 子句内加入多个条件，如：

```
SELECT   *
FROM     Northwnd
WHERE  ShipCountry = "Canada" AND(OrderDate > = '01/01/97') AND(OrderDate > = '01/31/97')
```

以上的条件表示寻找 ShipCountry 字段等于 Canada，且 OrderDate 字段值介于“01/01/97”和“01/31/97”之间的记录。若未特别设定，查询结果的排序会套用数据表的索引排序方式，若要在查询结果中的不同字段执行排序，可以使用 ORDER BY 子句，如以下的例子表示 OrderDate 字段使用的递增排序：

```
SELECT   *
```

东软电子出版社

```
FROM    Northwnd
WHERE  ShipCountry = "Canada" AND(OrderDate > = '01/01/97') AND(OrderDate > = '01/31/97')
ORDER BY OrderDate
```

排序的方式有递增和递减，若未特别声明，表示使用默认的递增排序，也可以指定 ASC 表示递增排序，DESC 表示递减排序，以下范例表示将最近的订单记录显示在查询结果最上方：

```
SELECT  *
FROM    Northwnd
WHERE  ShipCountry = "Canada" AND(OrderDate > = '01/01/97') AND(OrderDate > = '01/31/97')
ORDER BY OrderDate DESC
```

对于数据表中各项数字数据进行统计，也就是求和、求平均等计算，可以使用 GROUP BY 子句，再加上需要的计算方式，这些使用于 GROUP BY 子句的多种计算方式又称为聚合函数。

如以下的查询可以计算 ShipCountry 的数量，使用的聚合函数是 COUNT，表示以 GROUP BY 子句后的字段为依据，计算 COUNT 函数括号内字段不同值出现的次数。

```
SELECT ShipCountry , COUNT(ShipCountry) AS OrderCount
FROM Orders
GROUP BY ShipCountry
ORDER BY OrderCount DESC
```

结果会显示各个国家的出货次数统计，且由于在 ORDER BY 子句加上 OrderCount 的递减排序，所以显示结果的第一笔会是最大值，依次递减。

查询中 GROUP BY 子句代表计算的单位，所以本例结果 ShipCountry 字段值不会重复，因为每个国家只有一个统计结果，数据库引擎会扫描数据表的每一笔记录，并计算每个国家的出现次数。

本例中使用了 COUNT 函数，T-SQL 可以使用多个聚合函数，COUNT 是其中一个，以下是可以使用的聚合函数：

- Count：统计记录的笔数
- Sum：统计字段内数字数据的总和
- Avg：统计字段内数字数据的平均值
- Min：统计字段内数字数据的最小值
- Max：统计字段内数字数据的最大值

5. 更新 SQL 数据

若要更新现有的数据内容，可以使用 UPDATE 命令。这个命令可以针对现有记录的一个或多个字段，更改数据，如以下示例表示在 Customers 数据表内更改一个字段的数据：

```
UPDATE Customers
SET ContactName = "Maria Anderson"
WHERE CustomerId = "ALFKI"
```

上述范例的 UPDATE 命令后是将要更改内容的数据表名称，其后再使用 SET 关键字设定更改的字段及新数据，所以此处表示将 ContactName 字段更改为

东软电子出版社

MariaAnderson,但不是更改所有记录,因为本例中还使用了 WHERE 子句,所以结果是只更改 CustomerId 等于 ALFKI 的记录,由于 CustmoerId 是主索引,故上述语法必定只更改一笔记录。一个 UPDATE 命令之后,必须以 SET 关键字设定更改的字段及新值,且只能有一个 SET,其后可以设定多个字段,成为“SET 字段 1= 新值,字段 2=新值……”。更改记录数量的多少,视 WHERE 子句的定义而定,在主索引设定条件就只更改一笔,若不是主索引,就可以更改多笔记录,如以下示例表示将所有 Country 字段等于 USA 的记录的同一个字段改为 United States:

```
UPDATE Customers
SET ContactName = "United States"
WHERE CustomerId = "USA"
```

使用 UPDATE 指令务必设定 WHERE 子句,且条件必须有效,因为若未设定 WHERE 子句或条件无效,就会更改整个数据表的所有记录。以下是通过 UPDATE 命令中更改多个字段的范例:

```
UPDATE Customers
SET ContactName = "United States" , CITY = "Tokyo"
WHERE CustomerId = "USA"
```

6. 新增 SQL 数据

若要在数据表中新增记录,可以使用 INSERT 命令。INSERT 命令可以在指定数据表中新增记录,新增时必须指定数据表、字段及数据,如以下范例表示在 Order Details 数据表新增一条记录,且在五个字段中增加数据。

```
INSERT  INTO [Order Details]
(orderId , ProductId , UnitPrice , Quantity , Discount)
VALUES(10248 , 2, ,19.00 , 2, 0)
```

若数据表或字段名称含有空格,必须在名称前后加上中括号,才能识别。以上示例在数据表名称后,以小括号设定五个字段名称,其后是 VALUES 及又一对小括号,其内是分别对应第一对小括号内五个字段的数据。但不是所有字段都必须设定,若字段已有默认值、识别值或可以为 NULL,就可以不用设定,如:

```
INSERT  INTO [Order Details]
(orderId , ProductId , UnitPrice , Quantity )
VALUES(10248 , 2, ,19.00 , 2)
```

以上的 INSERT 命令使用四个字段,较前例少了 Discount 字段,甚至字段顺序也可以互换,如:

```
INSERT  INTO [Order Details]
(ProductId , orderId , UnitPrice , Quantity )
VALUES(10248 , 2, ,19.00 , 2)
```

以上三个 INSERT 命令都使用 VALUES 关键字,表示每次新增一条记录。也可以配合 SELECT 命令,新增多条记录,如以下示例会由 SUPPLIERS 数据表取出记录,再新增至 Products 数据表:

```
INSERT INTO Products
(SupplierId , ProductName , CategoryId)
SELECT SupplierId , "Almond" , 7
```

东软电子出版社

```
FROM Suppliers
```

以上的命令在执行时，会先以 SELECT 命令取得记录集，再逐笔加入至 INSERT INTO 后指定的数据表，而各个字段的数据类型也必须正确对应，如字符型的数据，不可新增数字型的值。

7. 删除 SQL 数据

若要在数据表中删除记录，可以使用 DELETE 命令。该命令可以在数据表内删除记录，以下的示例为了不影响现有的数据，所以先用 SELECT…INTO…命令，将 Customers 数据表复制成为 CustomersCopy 数据表，并在后者执行删除记录的动作：

```
SELECT * INTO CustomersCopy
FROM Customers
```

以上命令表示由 FROM 后的现有数据表，取出记录，并置于 INTO 后的新数据表，由于使用“*”字符，所以新旧数据表的结构及内容是完全相同的。

若要在 CustomersCopy 数据表删除一条记录，可以在 DELETE 指令之后加上 WHERE 子句，以主索引字段作为条件，如：

```
DELETE  FROM CustomersCopy
WHERE CustomerId = "ALFKI"
```

以上示例表示在 CustomersCopy 数据表删除 CustomerId 字段等于 ALFKI 的记录。使用 DELETE 命令的重点是务必加上条件，否则如下的指令，会删除 CustomersCopy 的所有记录：

```
DELETE FROM CustomerCopy
```

7.2.2　创建和访问存储过程

存储过程是存储在 SQL Server 数据库内的对象，它可以内含多行、多个 SQL 命令，在需要时予以执行，更重要的是存储过程可以作为 SQL Server 数据库与外部作业的接口。

以上的多个示例都是单行 SQL 语法，在需要时必须重新建立。而存储过程等于是 SQL 语法的存储位置，将常用执行动作存储为存储过程，就可以调用及执行，有点类似窗体应用系统的方法。

使用存储过程可以带来以下两个优势：一是可以将复杂而常用 SQL 语法予以集中存储，便于维护及调用；另是存储过程的执行速度会比查询快很多，能够有效的提高应用系统的执行效率。

1. 在 Visual Studio 中建立存储过程

在 T-SQL 中建立存储过程的命令是 CREATE PROCEDURE。该命令的功能是在指定的数据库内，建立新的存储过程。操作示例前，请先启动 Visual Studio。

(1)在服务器资源管理器中打开数据库节点，在“存储过程”节点上单击鼠标右键并选择“增加新存储过程”。

(2)在开启的代码编辑窗口中，输入如下代码内容：

```
ALTER PROCEDURE dbo.StoredProcedure1
AS
    SELECT Northwnd.* FROM Northwnd
```

东软电子出版社

```
    RETURN
```

(3)按下“Ctrl+S”键,执行存储,就会将存储过程保存至数据库。

(4)若要执行存储过程,方法一是开启“存储过程”节点,在欲执行的存储过程名称上单击鼠标右键,再选择“执行”,就会显示由存储过程回传的结果。

2. 建立有参数的存储过程

存储过程也可定义参数,执行时传入不同参数,可获得不同结果或在 SQL Server 执行时指定动作。参数可以使存储过程更加展现它的灵活性,因为执行时输入的参数值,可以作为存储过程内执行动作的依据。

(1)在服务器资源管理器打开数据库节点,在“存储过程”单击鼠标右键选择“增加新的存储过程”。

(2)在打开的代码编辑窗口输入以下内容:

```
ALTER PROCEDURE dbo.StoredProcedure1
    (
        @CustomerId char(5),
        @TotalSales money OUTPUT
    )
AS
    SELECT @TotalSales = SUM(Quantity * UnitPrice)
    From (Customers INNER JOIN Orders
    ON Customers.CustomerId = Orders.CustomerId)
    INNER JOIN [Order Details]
    ON Orders.OrderId = [Order Details].OrderId
    WHERE Customers.CustomerId = @CustomerId
    RETURN
```

(3)按下“Ctrl+S”键保存存储过程。

在本例建立的存储过程程序中,使用了两个参数,分别是@CustomerId 和@TotalSales。其中@CustomerId 是输入参数,表示在执行这个存储过程时,必须在此参数传入数据;@TotalSales 是输出参数,表示在存储过程执行完成后,传回值会放在此参数内。若在 Visual Studio 执行此存储过程,会显示如图 7-9 所示的对话窗口,要求输入参数值。

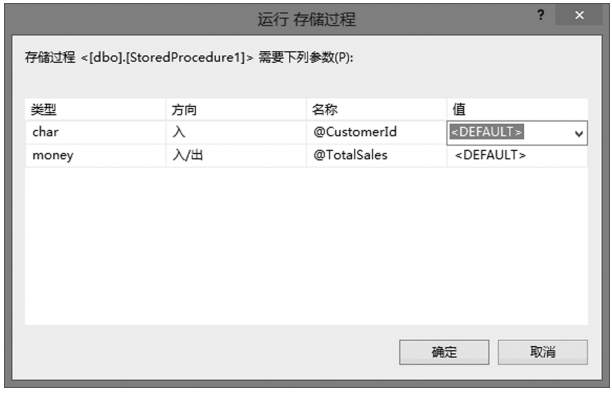

图 7-9 执行有参数的存储过程时显示的对话框

东软电子出版社

用户可在如图 7-9 所示窗口的第一个参数的“值”输入 ALFK1，第二个参数的“值”输入 0，再按下“确定”按钮，即可执行存储过程，结果会显示在 Visual Studio 的输出窗口中。

3. 在 C# 应用程序中执行有参数的存储过程

以下示例完成 C# 应用程序执行有参数的存储过程。

(1)建立新的 Widows Froms 应用项目，命名为 ParameterizedSP。

(2)在显示的 Form1 的表单上加入两个 Label 控件，分别命名为 Label 及 TotalSalesLabel。一个 TextBox 控件，命名为 CustomerIdTextBox，一个 Button 控件，命名为 GetTotalSalesButton。

(3)在 Button 控件的 Click 事件中添加如下示例代码：

```
private void GetTotalSalesButton_Click(object sender, EventArgs e)
    {
        TotalSalesLabel.Text = string.Format("Total Sales: {0}",
            GetTotalSales(CustomerIdTextBox.Text));
    }
```

(4)在代码编辑窗口的 Form1 类别加入名为 GetTotalSales 的方法，内容如下：

```
    private double GetTotalSales(string customerId)
    {
        double totalSales = -1;
        try
        {
            string connectionString = "Data Source = BTIC - TJ - 065;
                Initial Catalog = DataBaseTest;Integrated Security = True";
            SqlConnection connection = new SqlConnection(connectionString);
            SqlCommand command = connection.CreateCommand();
            command.CommandType = CommandType.StoredProcedure;
            command.CommandText = "StoredProcedure1";
            command.Parameters.AddWithValue("@CustomerId",customerId);
            command.Parameters.AddWithValue("@TotalSales",null);
            command.Parameters["@TotalSales"].DbType = DbType.Currency;
            command.Parameters["@TotalSales"].Direction = ParameterDirection.Output;
            connection.Open();
            command.ExecuteNonQuery();
            totalSales = Double.Parse(
                command.Parameters["@TotalSales"]
                .Value.ToString());
            connection.Close();
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        return totalSales;
```

东软电子出版社

```
    }
```

(5)在程序代码最上方使用 using,加入如下两个引用:

```
using System.Data;
using System.Data.SqlClient;
```

保存项目,执行项目,即可看到运行窗口。

本示例的目的是在 C#程序中执行有参数的存储过程,执行位置是 GetTotalSales 方法,方法内取得数据连接后,以 sqlCommand 对象代表存储过程,参数是使用 sqlParameter 对象,以两个 AddWithValue 方法加入至 Parameters 集合,且参数名称必须与存储过程的定义相同,第一个参数使用 customerId 变量的值填写,第二个参数以 Output 设定为输出,第一参数没有设定方向,默认是输入。

7.3 了解数据库连接方法

.NET Framework 平台对于普通文本文件、XML 及 in-memory 等三种数据,都提供经过最佳化处理的操作方式。如处理普通文本可用 System. IO 命名空间提供的各种类别。对于 XML 数据,可以使用位于 System. XML 命名空间的类别。in-memory 数据则可以使用 System. Data 命名空间所含类别。

7.3.1 连接普通文本文件

此类数据采用普通文本文件格式,可能是文字或任何可经识别的其他档案格式。

此类档案通常会以分隔符或固定长度的形式,形成半结构数据。例如以逗号或其他分隔符号,两个符号间的文字,视为一个字段的数据;固定长度则是以字符位置为依据,如第 1 至第 5 个字符,视为一个字段。在英文中称呼此类文档为 flat files,以与 XML 档案及数据库有所区别。

在关系型数据库出现之前,数据只能存储在一般文档内,所以普通文本档案是历史最悠久的存储方式,直到今天仍然会使用这种存储方式完成特定情况中的数据存储,最常见的是以 ini 为扩展名的文档。使用普通文本文件的好处是不同作业系统或不同软件都可以读取,常作为数据交换的选择。

针对档案的输入及输出作业,.NET Framework 平台是使用流(stream)及存储(backing store)的方式执行处理。流指的是数据传输过程,如同一连串的连续动作,backing store 则代表流两端的来源及目的,可以是各种存储形式的档案,这些不同的流处理方式,都可以使用在 System. IO 的相关类别。

可被视为数据库的普通文本文件,可以是纯文字或二进制格式。若是纯文字,通常横向一行的所有文字代表一条记录,行末标记代表记录结尾。可以使用 StreamReader 及 StreamWriter 类别针对纯文字文档执行读取及写入操作。

二进制文档的内容是“0”或“1”位元组,以位元组代表不同的数据,所以二进制文档无法直接阅读,通常使用二进制文档存储图片、声音、影像等。若要正确读取,必须在取得位元组数据后再加以适当解析,可以使用 BinaryReader 及 BinaryWriter 类别针对二进制文

东软电子出版社

档进行读取及写入操作。

以下范例将引导学员由 Customers 数据表取出部分数据，写入至文本文档，再由这个文本文档读取资料并输出至主控台窗口。

(1)建立新的“控制台应用程序”项目，命名为 WorkingWithTextFiles。

(2)在 Program 类别书写如下代码，并务必将变量 connectionString 的内容更改为目前环境的连接参数。

```
static void Main(string[] args)
{
    string myDocumentsPath =
        Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
    CopyDataToTextFile(myDocumentsPath
        + @"\CustomerList.txt");
    DisplayTextFile(myDocumentsPath
        + @"\CustomerList.txt");
}
static private void CopyDataToTextFile(
    string fileName)
{
    try
    {
        string connectionString = "Data Source = BTIC - TJ - 065;
            Initial Catalog = DataBaseTest;Integrated Security = True";
        SqlConnection connection = new SqlConnection(connectionString);
        SqlCommand command = connection.CreateCommand();
        command.CommandText =
            "SELECT CustomerId, CompanyName, "
            + "ContactName, Phone From Customers ";
        using (connection)
        {
            connection.Open();
            SqlDataReader reader =
                command.ExecuteReader();
            using (StreamWriter sw =
                new StreamWriter(fileName))
            {
                while (reader.Read())
                {
                    string customerRow =
                        string.Format("{0},{1},{2},{3}",
                        reader.GetValue(0),
                        reader.GetValue(1),
                        reader.GetValue(2),
                        reader.GetValue(3));
```

东软电子出版社

```
                    sw.WriteLine(customerRow);
                }
            }
        }
    }
        catch(Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
}
    static void DisplayTextFile(string fileName)
    {
        try
        {
            using (StreamReader sr = new StreamReader(fileName))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    Console.WriteLine(line);
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
```

(3)在程序的最上方用 using 进入如下两个应用：

```
using System.Data.SqlClient;
using System.IO;
```

(4)保存并执行项目，在“我的文件”中会生成 CustomerList.txt 档案，此文档内容应该来自 Customers 数据表四个字段的数据，最后再由这个文档读取数据及显示在主控台窗口。

在本例中写入操作使用 Stream Writer 类型的对象 sw，以此对象建立及写入指定文档，写入的动作是执行 WriteLine 方法，此方法每次会写入一行数据。读取文本文件的操作是使用 StreamReader 类型的对象 sr，读取在对象 sw 建立的文档，读取的动作是执行 ReadLine，即每次读取一行直到文档末尾，ReadLine 传回 null，表示已无资料可读取。

7.3.2 连接 XML 文件

XML(Extensible Markup Language)也是纯文本形式文档，但会有多个不同标记，代表不同的数据，通常 XML 会被视为半结构的数据。XML 也可以将数据存储为中继数据

东软电子出版社

的内容，如以下 XML 内容表示两笔客户记录：

```
<? xml version = "1.0" encoding = "utf - 8"? >
<! —Customer List - - >
<Customers>
    <Customer Id = "ALFKI">
        <CompanyName> Alfreds Futterkister </CompanyName>
        <Phone> 010 - 12345678</Phone>
    </Customer>
    <Customer Id = "EASTC">
            <CompanyName>Eastern Connection </CompanyName>
            <Phone> 010 - 5467342</Phone>
    </Customer>
</Customers>
```

使用 XML 的优点是即使不了解数据结构的设定，仍可由 XML 显示内容取得数据。关键是 XML 数据中的各种标记，也就是“<>”的内容，如同 HTML 中的标记一样，它们都是成对出现，表示结构定义的开始及结束。如<Customers>是此标记的起始位置，结束位置则是</Customers>，标记之内则是数据。

以上范例而言，第一行的如下内容是 XML 的宣告：

```
<? xml version = "1.0" encoding = "utf - 8"? >
```

以上内容的<? 称为处理命令或指引(processing instructions)，此处的指令是 xml，表示以下内容是一份 XML 格式的文件，接下来是 1.0 的版本编号以及 utf-8 的编码格式。

两个成对标记加上其内含的数据，称为元素(element)，如以下内容就是范例中的一个 XML 元素：

```
<Phone> 010-5467342</Phone>
```

以上内容的元素名称是 Phone，而其值是 010-5467342。XML 文件的元素也可以是巢状的，也就是元素内可以含有另一个元素，此时上下层的元素标记必须成对显示，不可混合。如在以下的错误示例中，CompanyName 及 Phone 元素的标记就没有依照上述规定置于正确的位置：

```
    <Customer Id = "EASTC">
        <CompanyName>Eastern Connection </Phone>
        <Phone> 010 - 5467342</CompanyName>
    </Customer>
```

所以 XML 文件内容是多层级的，每一个 XML 文件都会有一个单一最上层节点，在此节点之内再包含更多的子节点，且子节点也可以是多层级，亦可想像成树形结构形式。

元素(element)之内可以含有属性(attributes)，它是形容元素本身的规格或描述，如：

```
<Customer Id = "EASTC">
```

以上表示 Customer 元素含有一个属性，属性名称是 Id，而属性值是 EASTC。在 XML 中格式以“<! ——”开始，以“——>”结束的内容成为注解文字，这些文字不被视为 XML 内容，会在解析时被略过。

XML 几乎已成为目前公认的数据交换格式，所以许多工具都可以针对 XML 文件进

东软电子出版社

行读取及写入。.NET Freamswork 平台中有关 XML 的处理都在 System. Xml 命名空间中,在此命名空间使用的几个类别说明如下:

• XMLReader 及 XmlWriter:这两个类别可以针对 XML 文件,提供快捷、节省系统资源、只可向前处理(forward-only)的读取及写入操作。

• XmlDocument:这个类别可以在内存中处理 XML 数据,允许浏览及编辑等操作。

以下操作将完成读取 XML 文件的设计。

(1)建立新的"控制台应用程序"项目,命名为 WorkingWithXmlReader。

(2)将以下代码加入 Program 类别的 Main 方法中。

```
using (XmlReader reader = XmlReader.Create("Customers.xml"))
        {
            while (reader.Read())
            {
                switch (reader.Name)
                {
                    case "CompanyName":
                        if (reader.Read())
                        {
                            Console.Write("Company Name : {0} , ",
                                reader.Value);
                        }
                        break;
                    case "Phone":
                        if (reader.Read())
                        {
                            Console.WriteLine("Phone:{0}", reader.Value);
                        }
                        break;
                }
            }
        }
```

(3)在程序代码的最上方使用 using,加入以下引用:

```
using System.Xml
```

(4)在项目中新建或加入一个 XML,命名为 Customers. xml,文档内容如下:

```
<? xml version = "1.0" encoding = "utf - 8"? >
<! —Customer List - - >
<Customers>
    <Customer Id = "ALFKI">
        <CompanyName> Alfreds Futterkister </CompanyName>
        <Phone> 010 - 12345678</Phone>
    </Customer>
    <Customer Id = "EASTC">
        <CompanyName>Eastern Connection </CompanyName>
```

东软电子出版社

```
        <Phone> 010 - 5467342</Phone>
    </Customer>
</Customers>
```

(5)保存项目并将 Customer.xml 文档复制到项目文件夹中,再执行项目即可在主控台窗口显示来自 XML 文件的内容。

上述示例设计是使用 XmlReader 类别,在此类别执行 Create 方法,建立对象 reader,此对象即代表欲处理的 XML 文件,并请注意此文件必须是已存在的文档,若找不到文档会发生错误。接下来再以 Read 方法逐一读取 XML 文件各个节点,取得每一个节点的 Name 及 Value 属性,打印到主控制台窗口,直到读取完毕为止。

7.3.3　连接内存中对象

DataSet 是可以在内存存储不可见的数据库,因为都在内存中处理,故需要占用系统资源,但好处是与真实数据库是分离的,可以在有需要时再回传及存储。

DataSet 在内存的内容可以复杂到是一个关系型数据库,也可以只有一个数据表,如同在数据表软件中的设计。可以含有数据表、关联、数据整合相关设定等,这些设定可以在程序中逐步完成设计,也可以由现有数据库中取得,再放入 DataSet 内,放入之后就与数据库无关。所以 DataSet 无论如何产生,它的处理都只有在内存中进行,所以对 DataSet 的来源没有所谓的数据源或数据连接,因为它与物理存储中德数据库不会发生连接,只有在有需要时,才连接数据库将处理结果回传至数据库,或由数据库取得记录。所以 DataSet 的特性就是离线处理,即使网络不存在,仍然可以在内存中处理数据库,而在一个网络中,这种处理方式也可以减少网络传输负荷,因为只有在必要时才会通过网络连接至数据库。

处理 DataSet 的相关类别都在 System.Data 命名空间内,最重要的是 DataSet 类别。此类别处理内容就是一个可大可小的 DataSet,它的下层是一个或多个 DataTable,也就是数据表;DataTable 之下是 DataColumn 即字段,以上是结构定义的三个重要对象。记录则是 DataRow 对象,每一个 DataTable 对象都有 DataRow 集合及 DataColumns 集合,其内包含该 DataTable 含有的字段及记录对象。

在 DataSet 及数据库之间的中介是 DataAdapter 对象,这个对象可以接受数据库连接设定,它会连到指定数据库,执行指定动作,包括由数据库取得数据,或将 DataSet 中的数据回传至数据库。以下是.NET Framework 平台针对不同数据格式使用的多种 DataAdaptor 类别:

- OdbcDataAdpter:使用在以 ODBC 作为数据源时,此类别的位置是在 System.Data.Odbc 命名空间内。
- OleDbDataAdapter:使用在以 OLEDB 作为数据源时,此类别的位置在 System.Data.OleDb 命名空间内。
- SqlDataAdapter:使用在以 SQL Server 作为数据源时,此类别的位置是在 System.Data.SqlClient 命名空间内。

上述共有三种针对不同数据源的连接类别,若使用 SQL Server 数据库,建议使用 SqlDataAdapter,因为它是为 SQL Server 而设计,可以提供较好的处理效率。以下是针

东软电子出版社

对 DataSet 由建立至更新等操作的步骤：

(1)使用 DataAdapter 从数据源取得数据，放入至 DataSet 内的 DataTable 对象内。

(2)在各个 DataTable 对象内执行新增、更改、删除 DataRow 等操作。

(3)在 DataSet 执行 AcceptChanges 方法，表示在 DataTable 对象内的更改操作，这个方法的目的是变更 DataRow 对象的最新内容，因为在回存至数据库时，系统会比对每个 DataRow 对象的原始及最新内容，在判断是否要更新至数据库。也可以使用 RejectChanges 方法，表示拒绝变更，DataRow 对象就会恢复至变更前内容。

以下操作完成读取 DataSet 对象所含数据的设计。

(1)建立新的“控制台应用程序”，命名为 WorkingWithDataSet。

(2)在 Program 类别撰写如下程序代码，并务必将变量 cString 的内容，更改为当前环境的连接设置。

```
        static void Main(string[] args)
        {
            WorkingWithDataSet();
        }
        static void WorkingWithDataSet()
        {
             string cString = "Data Source = BTIC-TJ-065;Initial Catalog = DataBaseTest;
Integrated Security = True";
            SqlConnection northwindConnection = new SqlConnection(cString);
            string customerCommandText = "SELECT * FROM Customers";
            SqlDataAdapter customerAdapter = new SqlDataAdapter(customerCommandText,
northwindConnection);
            string ordersCommandText = "SELECT * FROM Orders";
            SqlDataAdapter ordersAdpter = new SqlDataAdapter(ordersCommandText,
northwindConnection);
            DataSet customerOrders = new DataSet();
            customerAdapter.Fill(customerOrders,"Customers");
            ordersAdpter.Fill(customerOrders,"Orders");
            DataRelation relation = customerOrders.Relations.Add("CustomerOrders",
customerOrders.Tables["Customers"]
                .Columns["CustomerID"],
                customerOrders.Tables["Orders"]
                .Columns["CustomerID"]);
            foreach (DataRow customerRow in customerOrders.Tables["Customers"].Rows)
            {
                Console.WriteLine(customerRow["CustomerID"]);
                foreach (DataRow orderRow in
                    customerRow.GetChildRows(relation))
                    Console.WriteLine("\t" + orderRow["OrderID"]);
            }
            Console.WriteLine("Press any key to continue...");
```

东软电子出版社

```
            Console.ReadKey();
        }
```

(3)在程序代码最上方使用 using,加入以下两个引用:

```
using System.Data;
using System.Data.SqlClient;
```

(4)保存并运行项目,就会在主控制台窗口显示来自 Customers 数据表的内容,且每一个 CustomerID 之后都会显示各个客户的 OrderID 数据。

在本例设计中首先建立一个 DataSet 对象,在此对象内建立两个 DataTable 及一个 DataRelation,表示建立两个数据表及关联,双方的关联字段是 CustomerID,以上是 DataSet 的结构定义。接下来以 CustomerAdapter 对象取得数据,并执行 Fill 方法,将数据填入至两个 DataTable,最后再以 foreach 循环,逐一取得 DataTable 的所有 DataRow 对象,也就是由 DataTable 取得记录。由于在定义时建立了 Datarelation,所以在内层循环使用 GetChildRow 方法,取得客户的子记录。这些子记录就是各个单一客户的订单的记录。

DataSet 对象内容也可以转换为 XML 的形式,如 WriteXml 方法可以将 DataSet 的内容写入至 XML 文件,若要将 XML 文件读取至 DataSet 内,需要执行 ReadXml 方法。

东软电子出版社