第一章 Delphi 开发数据库应用程序概述

在进行具体的数据库编程以前，有必要了解一些数据库的基本概念、功能特点和组织结构特点。

用户通过数据库应用程序可以与存储在数据库中的数据进行交互操作。数据库提供用来存储数据的特定结构，并且通过一定的机制实现多个应用程序对数据的共享。Delphi 对关系型数据库提供全面的支持。而关系型数据库将数据组织成相应的表，表中的行对应着相应的每一个记录，表中的列对应着数据库的字段。通过使用一些简单的操作就能实现对数据的控制和管理。

在设计一个数据库应用程序前，必须知道数据是怎样进行组织的。基于确定的数据库结构可以设计一个用户接口，通过这个接口可以显示用户需要的数据、接收用户输入的数据。这里说的用户接口实际上也是一个应用程序。设计一个数据库应用程序包括以下三部分的内容，这也是本章要介绍的内容：

1. 使用数据库；
2. 数据库结构；
3. 设计用户接口。

另外，本章还要介绍 Delphi 4.0 新增的一些技术，以及应用 Delphi 4.0 开发数据库应用程序的一般步骤。

1.1 使用数据库

使用数据库是数据库应用程序获得数据的来源，这就需要解决两个问题：首先必须知道在 Delphi 中运用什么部件访问数据库，即获得数据库中详细的数据信息；然后必须知道 Delphi 支持访问的数据库种类，这样才能对数据访问部件相应属性的设置。

1.1.1 数据访问部件(Data Access)

数据库应用程序必须与数据库进行数据交换，因此必须建立与数据库的联系。在 Delphi 中，这个工作是通过数据访问部件(Data Access)来完成的。这些部件使用 BDE(Borland Database Engine 的简称，即 Borland 数据库引擎)来访问数据库，并使其中的数据信息可以为用户接口中的数据控制部件所使用。

Delphi 提供一个数据访问部件的页，专门用于存放数据访问部件。随着 Delphi 版本的不断更新，数据访问部件页上的部件也不断增加。Delphi 4.0 中数据访问部件页(Data Access

Page）上的部件如图 1-1 所示。

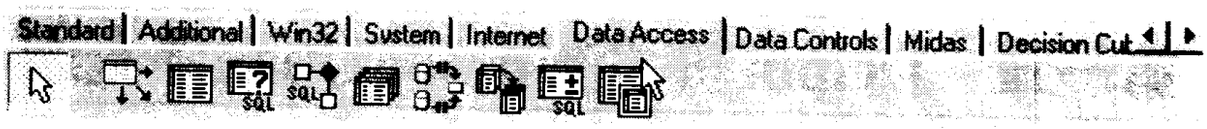

图 1-1　Delphi 的数据访问部件页

数据访问部件页上提供了一组数据访问部件用来访问数据库中的数据，如图 1-1 所示。

当要创建一个数据库应用程序时，首先在窗体中选择一个数据访问部件，然后为数据访问部件设置有关的属性，说明要访问的数据库、数据表以及表中的记录等，数据访问部件为数据控制部件与数据源建立一条通道。数据访问部件在程序运行时是不可见的。表 1-1 列出了数据访问页上的数据访问部件以及它们的主要用途：

表 1-1　　**数据访问部件列表**

数据访问部件名称	主要用途
TDataSource	作为数据访问部件 TTable、TQuery、StoredProc 与数据控制部件 TDBGrid、TDBEdit 等之间传送数据的通道
TTable	通过 BDE 存取数据库表中的数据，TTable 再与 TDataSource 进行连接，使得数据控制部件能够有效地从 TTable 中访问数据并能显示和编辑其中的数据
TQuery	它利用 SQL 语言访问数据库表中的数据，并与 TDataSource 进行，实现数据浏览部件对数据库的访问
TStoredProc	一般主要用来访问远程服务器中的存贮过程
TDatabase	当应用程序要登录到一个远程服务器上的数据库时，可以用该部件来建立应用程序与数据库永久性的连接
TBatchMove	用于复制数据库表的结构或表中的记录

在绝大多数数据库应用程序中：一般都是使用 TTable、TQuery 或 TstoredProc 等部件与数据库进行连接，用 TDataSource 部件连接数据控制部件和数据集部件。在数据库应用程序当中，TTable、TQuery 和 TstoredProc 等部件中都包含一个不可见的 TField 类型的对象 Fields，Fields 是一个串列表，它对应于数据库表或一个查询结果的列或字段。Fields 对象是伴随着 TTable、TQuery 和 TStoredproc 部件的活动状态动态地建立的。当数据库表被关闭时，Fields 对象也随之消失，它在程序设计和程序运行过程中都是不可见的。

当然也可以利用 Fields Editor 建立永久性的 Fields 对象供 Delphi 应用程序使用，在后

面的内容中将进行详细阐述。

1．TTable 部件

利用 TTable 部件程序设计人员甚至可以不需要编写任何程序便可对数据库进行访问，在一个应用程序窗体中放置一个 TTable 部件的过程如下：

首先在部件选择板上选择 Data Access 页，然后单击 Table 图标，接下来在窗体内单击鼠标，获得一个 TTable 部件，最后为 TTable 部件设置有关的属性。

2．TQuery 部件

TQuery 部件是使用 SQL 语言开发数据库应用程序的有力工具，因为使用 SQL 语言可以非常方便灵活地对一个或多个数据库表中的记录进行访问。利用 TQuery 可以查询本地的数据库如 Pararadox 和 dBASE 数据库系统中的数据，还可以使用 TQuery 部件对一个远地的数据库 SQL 服务器进行访问，建立 Client/Server 模式的应用程序。

在一个应用程序窗体中放置一个 TQuery 部件的过程如下：首先在部件选择板上选择 Data Access 页，然后单击 Query 图标，在窗体内单击鼠标，获得一个 TQuery 部件，接下来为 TQuery 部件设置有关的属性。

在这里要注意在 TQuery 部件中，不是用 TableName 属性来指定要访问的数据库中的数据库表，而是在 SQL 属性中，通过 SQL 语句来指定将要访问的数据库表。

3．TDataSouece 部件

TDataSource 部件是连接部件 TTable、TQuery、TStoredProc 和数据控制部件 TDBGrid、TDBEdit 等的桥梁。TTable、TQuery、TStoredProc 部件通过 BDE 可以实现与数据库的连接，但它们本身不能显示数据库中的数据信息。

数据控制部件如 TDBGrid、TDBEdit 等能够提供可视化的界面，显示数据库中的数据信息，但它们不具备访问数据库的能力。正是 TDataSource 将这两者有机地结合起来，使得用户才能交互地对数据库中的数据信息进行查询、修改、插入、删除等操作。

在应用程序窗体中放置 TDataSource 部件的过程如下：首先在部件选择板上选择 Data Access 页，然后单击 DataSource 图标，在窗体内单击鼠标，获得一个 TDataSource 部件，最后为 TDataSource 部件设置有关的属性。

1.1.2 Delphi 支持的数据库种类

根据使用 Delphi 的版本，BDE 中包含所支持数据库的驱动程序。在数据库应用程序中可以连接到相应的数据库上，这里的数据库种类又可以分为两种。一种是本地数据库，如 Paradox、dBASE、FoxPro、和 Access 等，一般用于编写单层的数据库应用程序，即本地数据库应用程序。另一种是远程数据库服务器，如 Interbase、Oracle、Sybase、Informix、Microsoft SQL server 和 DB2 等，一般用于编写双层数据库应用程序(即客户/服务器数据库应用程序)和多层数据库应用程序。有关双层数据库应用程序和多层数据库应用程序的内容在后面将进行详细介绍。

Delphi 数据库应用程序是通过 BDE 获取它们所需的数据的，BDE 与不同类型的数据源打交道，BDE 可以使用的数据源即 Delphi 支持的所有数据库种类有如表 1-2 所示。

表 1-2　　Delphi 支持的数据库种类

数据源(DataSource)	特性描述
DBASE 数据库(.DBF)	数据库表是通过 dBASE 数据库管理系统或 DBD 建立的，每个表是一个独立的文件
Paradox 数据库(.DB)	数据库表是通过 Paradox 数据库管理系统或 DBD 建立的，每个表是一个独立的文件
ASCII 文件(.TXT)	表是通过 Database Desktop 建立的，每个表是一个独立的文件
本地 InterBase 服务器(.GDB)	数据库是通过 InterBase 数据库管理系统建立的，多个表包含在一个数据库文件中
SQL 数据库服务器(ORACLE、Sybase、Informix 、Microsoft SQL Server、InterBase)	数据库是通过相应的数据库服务器提供的专用或通用工具建立的，也可以通过 DBD 来创建数据库，并通过 SQL Link 访问数据库

1.2　数据库结构

用户获得的数据以及进行存储的数据在数据库中都是根据一定的组织形式保存的。现在采用数据库种类一般是关系型数据库。根据使用不同关系型数据库的种类，数据库的结构都有各自的特点，但在功能实现上基本一致。

关系型数据库（Relational Database）：一个关系型数据库是由若干表组成。在 Delphi 中，数据库概念对应到物理文件上是有一些不同的。对于 dBASE、FoxPro、Paradox 这三种数据库系统，数据库对应于某一个子目录，而其它类型如 MS Access、Btrieve 则是指某个文件。这是因为前者的表为单独的文件，而后者的表是聚集在一个数据库文件中的。

表（Table）：一个表就是一组相关的数据按行排列，象一张表格一样。比如一个班所有学生的期末考试成绩，存在一个表中，每一行对应一名学生，在这一行中，包括学生的学号、姓名以及各门课程的成绩。

字段（Field）：在表中，每一列称为一个字段。每一个字段都有相应的描述信息，如数据类型、数据宽度等。

记录（Record）：在表中，每一行称为一条记录。

索引（Index）：为了加快访问数据库的速度，许多数据库都使用索引。

1.3　设计用户接口

用户接口是技术上的说法，说得通俗一点就是数据显示界面和数据更新界面。Delphi

提供一组数据控制部件，用于显示数据库中各条记录的各个字段，接收用户对数据的各种更新。

在数据控制部件页（Data Control Page）上的部件用来与用户交互，显示、修改数据库中的数据，如图 1-2 所示。

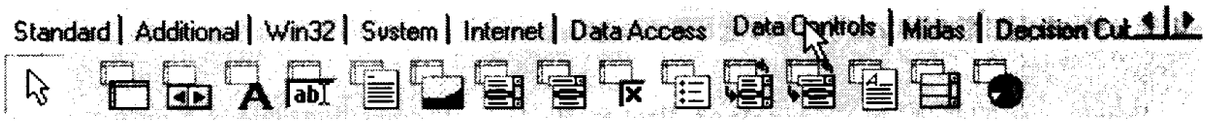

图 1-2 Delphi 的数据控制部件页

数据控制部件页上的部件(如图 1-2 所示)，主要用于设计用户界面，对数据库中的数据进行浏览、编辑、插入、删除等操作。因而数据控制部件常常又被称为数据浏览部件，数据控制部件其实是在 Standard 页上的标准部件的基础上，相应地增加了数据浏览功能，使得它们能够显示和编辑数据库中数据信息。

数据控制部件既能够把数据库中的数据显示到窗体中，又可以将其自身的经过修改的数据写回到数据库中。数据控制部件为开发 Delphi 数据库应用程序提供可视化的用户界面，不管应用程序是访问本地数据库中的数据文件，还是访问远程数据库服务器中的数据文件，用户界面都是一致的，即数据库的物理位置对数据控制部件是透明的。

有关数据控制部件的内容将在后面的章节进行详细的介绍。

1.4 Delphi 4.0 在数据库方面的新技术

Delphi 4.0 的数据库方面的新技术大致表现在以下四个方面：Web 上的 Client/Server 技术、数据分析、企业组件和提高程序设计人员的生产力。

有关数据库的处理技术已经发展了很长时间，先后出现过三种数据库系统：

第一代数据库系统采用单层结构(1-Tier)，这种结构很浪费计算机资源。第二代数据库采用了 Client/Server 结构，也就是所谓的两层结构(2-Tier)。这种结构得到了广泛的应用并获得了极大的成功。但在其结构上也存在一些问题，主要表现在应用程序的伸缩和维护方面。例如，在网络上如何维护数据的统一性和完整性；一旦应用程序有任何改动，维护人员就必须对每个客户端进行修改。

最新一代的数据库系统是在传统的 Client/Server 结构中增加一个应用服务器，这种新结构叫做 n-Tier 或 Multi-Tier。

Delphi 4.0 针对这种新一代的数据库管理系统观念，提出了三种 Broker 和新一代的数据库引擎，来适应 n-Tier 的需求。

第一种叫做 Remote Data Broker。其结构的精髓是让每一个客户端不再需要 BDE，取而代之的是一个中央化的 BDE，以集中管理的方式降低每一个客户在 BDE 上调整的开销和复杂度。

第二种叫做 Constraint Broker。顾名思义，它所扮演的角色就是保证所有客户数据的

一致性及数据的完整性。

第三种是 Business Object Broker。它的目的是给一些关键性的商业应用程序提供一个快速且可信赖的使用环境。为了达成这种高层次的要求，Business Object Broker 自动地将应用程序做适当的划分，并复制重要的业务规则到每一个区间，以达到速度的要求。

1.4.1 Web 上的 Client/Server 技术

Delphi 4.0 的两项新技术——Web Broker 和 ActiveForms 可以使开发人员结合原有的 Client/Server 技术开发出 Web-Enabled 的 Client/Server 应用程序。

我们先从实际的层面来看，当一个 Client/Server 结构的数据库程序改成 Web-Enabled 的时候，首先要考虑的是如何把结果显示在不同的机器上，其次是如何将已经开发出来的程序方便地分发给不同地方的用户。

Delphi 4.0 中 Web Broker 的目的就是让服务器执行的结果以 HTML 格式快速地显示在 Web 浏览器上面。Web Broker 中包含了几个重要的核心成分：

WebServer Application：建立 ISAPI/NSAPI 或 CGI 的 DLL，或者建立执行程序。

WebBridge：让开发人员开发一些可在 ISAPI/NSAPI 中共用的 API，使得程序的开发不会因 Netscape 或 Microsoft 的标准改变而受重大的影响。

WebModeules：WebModule 所扮演的角色就好像是 WebServer 的信息中心，它分派每个使用者的要求，定义 URI 和新建 HTML 网页。

Sessions：Session 可同时处理许多数据库处理的要求，针对每一个用户请求，在中央的 BDE 中新建不同的 BDE 区段来应答用户。

WebDispatcher：是一个处理数据库需求的中心，允许 Delphi 的开发者把传统的 Client/Server 数据库所产生的内容传送到 Web 上去。

HTML Producers：主要的目的是将数据库信息或其它程序的结果转换为 HTML 的格式。

ActiveForm：是开发 Web-Enabled 的 Client/Server 应用程序另外一个重要的部分，它针对以 Windows 为基础的 Client/Server 结构的数据库系统。

Delphi 4.0 提供了一种最容易的方式，将 Delphi 的 Form 转成 ActiveForm。ActiveForm 是一个 ActiveX 控件，它用 Delphi 的 Form 为载体来装其他 Delphi 的组件，为了适应网络频宽不足的限制，ActiveForm 配合 Remote Data Broker 可以产生小巧且无须编程的 Multi-Tier 数据库应用程序。

Delphi 4.0 另一个有力的工具叫 Web Deployment：它可以自动地产生 HTML 文件，传送 ActiveForm 和其他所必需的文件到客户端。Web Deployment 可以支持 CAB 的文件格式，以增加文件下传的速度。除此之外，它还提供 Code Signing 机制去提高网络安全性。

Delphi 4.0 还提供了新的编译技术——VCL Package 技术，它可以把 Delphi 的应用程序缩小到 15K 左右，非常适合网络传输。

专家指导

什么是 VCL Package 技术呢?在 Delphi 4.0 里这种新编译技术大量运用在新兴的 Multi-Tier 数据库应用程序上，这种技术让程序开发者下面运用在新兴的 Multi-Tier 数据库应用程序上。这种技术让程序开发者轻易地将自己的应用程序拆解成小的程序片段。这可和一般的工具程序不同，因为每一片段都已经被独立编译过了，并且可重复使用，其原有程序的执行效率不会因拆解而降低。

1.4.2 数据分析

Delphi 4.0 另外值得一提的部分是在客户端的开发应用，主要表现在三个方面：数据分析，组件制造中心和减小程序开发的难度。

正确的数据分析，在业务决策分析上是相当的重要。当大量的数据放在用户面前时，用户要搞清这些繁杂数据背后的真实意义可不是件容易的事。如何将现有的数据转成有用的信息，来辅助决策者做出最正确的决择呢?一般来说，运用图形及报表形式是最好不过的了，Delphi 4.0 提供了三个重要的组件，Decision Cube，TeeCharts 和 Quick Report。

Decision Cube 让使用者能依据个人需求，动态查询数据，显示多维数据和动态图表，产生不同性质的有用分析结果，充分显示数据背后的相互关系，提供给快策者最快最有用的信息，帮助他作出最正确的决定。TeeCharts 是一个三维的可视化图表，使数据更容易被理解。Quick Report 是一个超强的报表生成组件群，除了提供一般的报表外，还可以产生 HTML 的报表格式，让所产生的报表得以在 Web 上发布。

1.4.3 组件制造中心

企业组件的制作基础是运用 ActiveX 的技术，配合对 COM 的支持能力，以一步到位 ActiveX 的工具，建立企业组件，提供企业内组件的最大重用能力。用 Delphi 4.0 制造出来的组件可以运用于各种不同的开发工具中，例如 C++Builder、IntraBuilder、VB 等。

1.4.4 提高程序设计人员的生产力

在程序开发过程中，语言的语法、组件的方法和事件的名称，或已经声明过的变量、过程的名称往往很难准确地书写。Delphi 4.0 提供了许多辅助方法来解决这些问题，减少程序开发所需时间。在 Delphi 4.0 有三个 Wizard，分别是 Code Template Wizard，Code Completion Wizard，Code Parameter Wizard 来帮助开发者避免一些在程序开发时不必要的资源和时间浪费。

1.5　运用 Delphi 4.0 开发数据库应用程序的步骤

在详细介绍开发步骤以前，有必要介绍 Delphi 4.0 中的数据库窗体向导(Database Form Wizard)。因为随着各种编程技术的不断发展，它们的智能化水平越来越高，这也是将来编程技术发展的一个重点。

1.5.1　Database Form Wizard

Delphi 为用户开发简单的数据库应用程序提供了一个开发工具叫做“数据库窗体向导”（Database Form Wizard），在 Delphi 系统菜单 Database 菜单下可以找到，如图 1-3 所示。

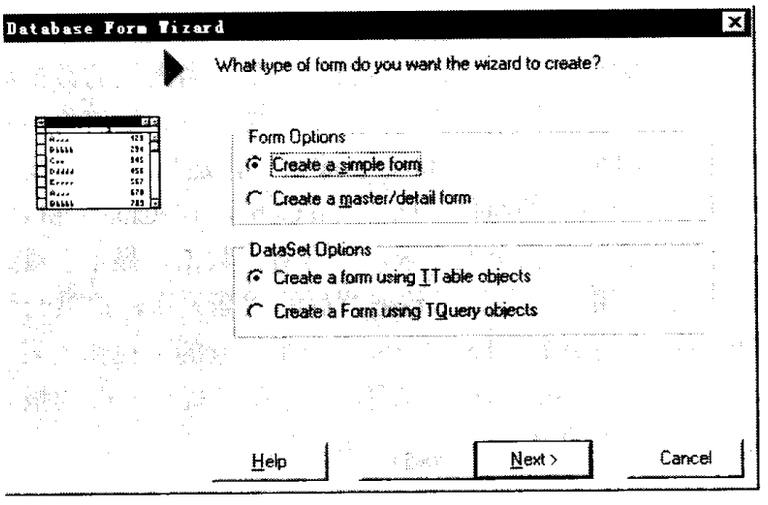

图 1-3　数据库窗体向导

数据库窗体向导能够自动生成简单的数据库应用程序中所必须完成的许多任务，它还可以生成基于单个数据库表的应用程序窗体或基于主要——明细型多个数据库表的应用程序窗体，数据库窗体向导能够自动完成的任务如下：

- 放置数据库部件到窗体中（TDataSource 部件）；
- 为数据集部件（TTable、TQuery）和磁盘上的数据库建立连接；
- 建立数据源（TDataSource）与数据控制部件的连接，数据源（TDataSource）与数据访问部件（TTable、TQuery）的连接；
- 为 TQuery 部件编写 SQL 语句；
- 为窗体中的部件定义 Tab 顺序。

1.5.2　开发数据库应用程序的基本步骤

数据库应用程序的一般开发的三个基本步骤如下：

程序设计；

功能实现；

程序运行和维护。

在这三个基本步骤中，都包含着数据库的开发和应用程序界面的设计两大类任务。对于一个客户/服务器模式的数据库应用程序，数据库和应用界面的区别就更明显一些，因为它们运行在不同的平台之上，而且使用的操作系统都常常不一样，如一个 Unix 环境的服务器和 Windows 环境的客户机。

1.5.2.1 程序设计

程序设计阶段应当根据用户的需求，明确描述数据库（数据库服务器端）和应用程序界面（客户机端）实现的功能。即决定哪些功能由服务器端实现，哪些功能由客户机端实现，对于客户/服务器应用程序，许多功能既可以在服务器端实现又可以在客户端实现的。

1.5.2.2 功能实现

功能实现阶段的主要任务是使用 Delphi 提供的工具和部件以及 Pascal 语言实现系统设计阶段的设想，并进行调试。

在功能实现阶段，最好使用数据库的一个备份数据库，这个备份的数据库与原数据库具有相同的结构，但其中的数据库只是原数据库中的一部分。之所以不在原数据库上开发应用程序，是因为考虑到没有调试好的应用程序可能会破坏数据库中的数据或者妨碍数据库的正常操作。

1.5.2.3 程序运行和维护

一个应用系统性能的优劣，效率的高低始终应当由用户来做出判决。应用程序在运行过程中，用户会提出一些新的需求和建议，根据用户需求的变化，应当对应用程序做一定的修改，使其进一步地得到完善和提高。

1.5.3 形成完整的数据库系统

选择好适当的数据库种类，并设计好数据库应用程序，这样就形成了一个完整的数据库系统。这也是编写数据库应用程序的目的，编写完的数据库应用程序并不是孤立的，它和数据库、数据库管理系统构成一个完整的数据库系统的各个部分,如图 1-4 所示。

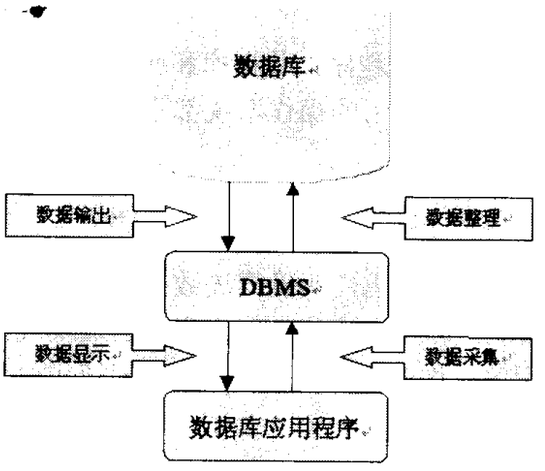

图 1-4　数据库系统的基本结构

数据库系统提供了一种把信息集合在一起并进行存储和维护的方法。如图 1-4 所示，数据库系统主要由三大部分组成：数据库管理系统（DBMS：它是专门负责组织和管理数据信息的程序）、数据库应用程序（它使我们能够获取、显示和更新由 DBMS 存储的数据）、数据库（按一定结构组织在一起的相关数据的集合）。

数据库管理系统（DBMS）是用于描述、管理和维护数据库的程序系统，是数据库系统的核心组成部分。它建立在操作系统的基础上，对数据库进行统一的管理和控制。其主要功能有：描述数据库的逻辑结构、存储结构、语义信息和保密要求等；管理数据库，控制整个数据库系统的运行，控制用户的并发性访问，检验数据的安全、保密与完整性，执行数据检索、插入、删除、修改等操作；维护数据库，控制数据库初始数据的装入、记录工作日志、监视数据库性能、修改更新数据库、重新组织数据库、恢复出现故障的数据库；数据通信以及组织数据的传输。

数据库种类在确定后，就决定了数据库应用程序的各项设置。

DBMS 中存储了大量的数据信息，其目的是为用户提供数据信息服务，而数据库应用程序正是与 DBMS 进行通信，并访问 DBMS 中的数据，它是 DBMS 实现其对外提供数据信息服务这一目的的唯一途径。简单地说，数据库应用程序是一个允许用户插入、修改、删除并报告数据库中的数据的计算机程序。

小结

本章的内容正如标题所示，主要概述怎样利用 Delphi 开发数据库应用程序。这章在全书中起到一个总纲的作用，将在 Delphi 数据库开发过程涉及的一些内容展现在读者的面前，便于后面内容的进行。

本章首先介绍设计数据库应用程序的三部分主要内容：怎样使用数据库、了解数据库的结构和怎样设计用户接口。接下来介绍 Delphi 4.0 在数据库方面新增的一些技术。最后，数据库应用程序程序和数据库、数据库管理系统形成一个完整的数据库系统。

第二章 数据访问部件(Data Access)

数据访问部件(即 Data Access)是 Delphi 中一组数据库存取部件，连接着数据库和数据控制部件。数据访问部件是 Delphi 中最重要的部件，数据库应用程序只有通过它才能获得数据库中的数据信息。本章将对数据访问部件组中的各个部件进行介绍，在内容上这样进行安排：

1. 数据访问部件简介及层次结构；
2. 数据集部件介绍，包括 Table、Query、StoreProc 部件；
3. DataSource 部件介绍；
4. NestedTable 部件介绍；
5. Database 部件介绍；
6. Session 部件介绍；
7. BatchMove 部件介绍；
8. UpdateSQL 部件介绍。

2.1 数据访问部件简介

在通常的数据库应用程序开发中，一般只使用数据访问部件中的 DataSource、Table 和 Query 部件就能满足绝大部分开发的要求。在开发客户/服务器结构的数据库应用程序，如果要进行事务处理，还需要使用 DataBase 部件。其它的数据访问部件对于扩大应用程序的功能也有很大的作用。这章将对数据访问部件组中的部件进行详细介绍，Delphi 提供的数据访问部件页如图 2-1 所示。

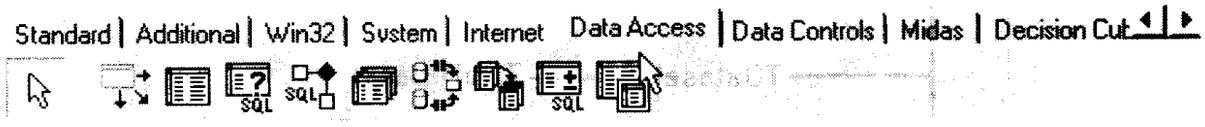

图 2-1 数据访问部件页

数据集部件(DataSet)，包括 Table、Query 和 StoreProc 部件。这种数据集部件直接用于连接到数据库。Table 部件通过 BDE 从实际的数据库表中获得数据，并且通过 DataSource 部件提供给数据控制部件；然后还能通过 BDE 将从用户获得的数据提交给数据库。Query 部件专门用于应用程序中涉及 SQL 编程的部分，Query 部件支持使用 SQL 语句通过 BDE 从实际的数据库表获得数据，并且可以通过 DataSource 部件将数据提供给数据控制部件；

同样 Query 部件支持使用 SQL 语句通过 BDE 从用户获得数据，并将数据提交数据库。StoredProc 部件使得一个应用程序可以进入到服务器的存储过程，并且还能够将从用户获得的数据通过 BDE 提交给数据库。

DataSource 部件建立起从数据集部件到数据控制部件之间数据交换的通道。在数据库应用程序中这个部件是不可见的，但又是不可缺少的。

NestedTable 部件用于从一个嵌套的数据库中获得数据，并可以通过一个 DataSource 部件将数据提供给数据控制部件。

Database 部件用于建立从数据库应用程序到数据库的永久性连接，一般用于具有远程数据库服务器的客户/服务器结构的数据库应用程序，特别是需要用户登录的远程数据库服务器。

Session 部件对数据库应用程序提供全方位控制。每个数据库应用程序都将产生一个缺省的 Session 部件，这个部件用于实现比较复杂的功能。比如要编写一个多线程的数据库应用程序，就需要对 Session 部件进行编程，并且每个线程对应着一个特定的 Session 部件。

BatchMove 部件用于复制一个数据库表的结构或者数据，支持将一个数据库中所有数据库表转移到另一种形式的数据库。

UpdateSQL 部件允许程序设计者使用 Delphi 缓存数据更新的特性，这种特性专门用于只读的数据库。通过设置数据集部件的 UpdateObject 属性可以将 UpdateSQL 部件连接到数据库。当缓存的数据更新需要提交时，数据集将自动使用 UpdateSQL 部件。

Delphi 提供了强大的开发数据库应用程序的能力，正是基于这些向用户提供的大量数据访问部件。在这些部件中间，有些部件继承了另一些部件的属性、方法和事件，也就是说多部件之间存在着继承和被继承的关系。各部件的这种关联便构成了一个层次结构，如图 2-2 所示：

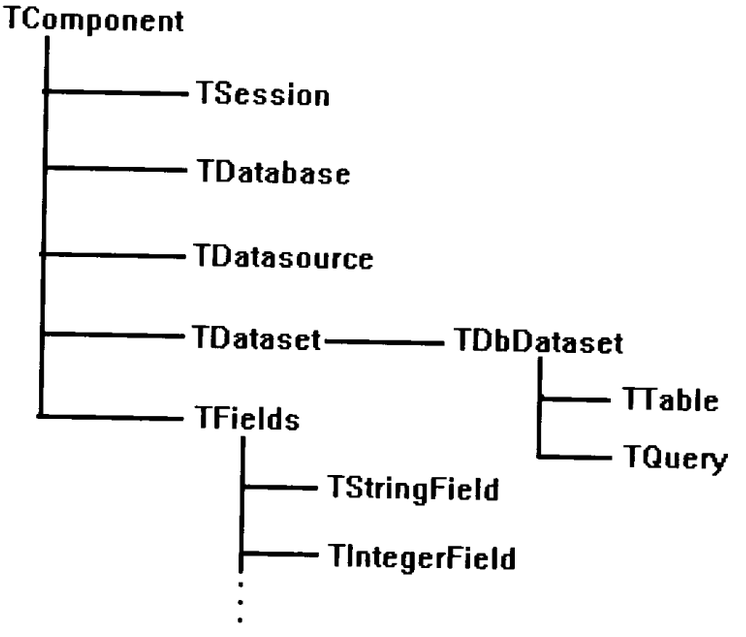

图 2-2　Delphi 数据访问部件的层次结构

图中的 Fields 部件对应于数据库表中的实际字段。它既可以在应用程序的运行过程中动态的生成，也可以在程序设计阶段通过字段编辑器来创建。它是不可见的部件，在程序中可以通过 TField 部件来访问数据库记录的各个字段值。

2.2 数据集部件(DataSet)

数据集部件建立了到数据库的连接，在数据库应用程序中就代表着数据库。这样在数据库应用程序中数据控制部件对数据的各种操作可以看作是对数据集对象的操作。这里将首先介绍数据集部件 Table，然后介绍数据集的有关内容。对数据集部件 Query 和 StoreProc 的介绍在后面 SQL 编程和客户/服务器程序编写的章节中将进行详细介绍。

2.2.1 数据集部件 Table

Table 部件是 Delphi 数据库编程中要经常使用的最重要的部件之一，它是数据库应用程序访问数据库时必须使用的数据集部件之一。在这一节里，将重点介绍 Table 部件特有的属性和方法。TTable 部件所有的属性、方法和事件都可以在联机帮助中查到。

2.2.1.1 TTabel 部件主要的属性

1.DatabaseName 属性

DatabaseName 属性是说明数据库应用程序所操作的数据库的名字，它可以是由 BDE 定义的数据库的别名、显式说明的数据库文件所在的路径或者是由 TDatabase 部件定义的一个数据库名。DatabaseName 属性常常是一个由 BDE 定义的数据库的别名。使用由 BDE 定义的数据库的别名代替数据库实际所在的路径和名字，好处是当实际的数据库存放的位置发生变化时，只需利用 BDE 简单的设置一下该数据库的别名，而数据库应用程序无需修改。

2.TableName 属性

TabelName 属性用以说明当前 TTable 部件所连接的实际的数据库表。DatabaseName 和 TableName 这两个属性一般都在设计阶段指定，当然在程序运行过程中也可以设置。但是要修改这两个属性时，要注意修改后 TTabel 的 Active 属性将自动变为 False。所以这时需要将 TTable 的 Active 属性改为 True，以免在程序运行时出错。

3.TableType 属性

TableType 属性说明与 TTable 部件相连接的数据库表的类型。当 TableType 属性设置成 Default 时，该属性所说明的数据库表的类型由数据库文件的扩展名决定。

若数据库文件的扩展名为.DB 或没有扩展名，表的类型是 Paradox 表；若数据库文件的扩展名为.DBF 时，表的类型是 dBASE 表；若数据库文件的扩展名为.TXT 时，表的类型是 ASCII 表。

如果 TableType 属性不设定为 Default，那么与 TTable 部件相连的数据库表的类型由 TableType 中的设置的值决定，不用考虑数据库文件的扩展名。

4.KeyExclusive 属性

KeyExclusive 属性的一个作用是说明在数据库表中查找记录时，将记录移到与查找值相匹配的记录处，还是将记录指针移到与查找值相匹配的记录后面一条记录处。该属性是布尔型变量，当它的值为 False 时(缺省情况下为 False)，将记录指针移到相匹配的记录处；为 True 时，将记录指针移到相匹配记录的后面一条记录处。该属性另一个作用是在表中指定检索范围时，用来说明是否包括满足过滤条件的边界记录。当 KeyExclusive 的值为 False 时，检索范围包括边界记录，否则不包括边界记录。有关详细的操作可以参看帮助中的“限定表中记录的检索范围”。

5.IndexFields 属性

IndexFields 的属性值是数据库表中字段名列表。IndexFields 包含与 TTable 部件相连的数据库表中的全部索引字段。

6.IndexFieldsCount 属性

IndexFieldsCount 属性说明表中索引字段的个数。这两个属性值都是只读的，只有在程序运行过程中可用。

7.IndexName 属性

IndexName 属性中存放着在建立数据库表时为数据库表定义的所有辅助索引名，它是一个辅助索引名列表，是只读属性。

8.IndexFieldNames 属性

IndexFieldNames 属性指定用于数据库表索引排序的字段名，多个字段名之间用分号隔开。例如对 Customer.DB 表中的客户记录按邮政编码 ZipCode 和客户号码 CustNo 排序时，可以设定 IndexFieldNames 的值为：ZipCode ; CustNo。

在 IndexFieldNames 属性中指定的字段必须存在于相应的数据库表中，否则会导致错误。IndexName 和 IndexFieldName 是互斥的，每次只能指定其中一个属性的值，不能同时为两个属性都指定属性值。

9.Exclusive 属性

Exclusive 属性是一个布尔型属性，它标明是否以共享方式打开数据库表。如果 Exclusive 的值为 True，当打开一个数据库表时，其他用户就不能访问该表了；若 Exclusive 的值为 False，将以共享方式打开一个数据库表。显然不能将其他用户正在访问的表以互斥方式打开(设定 Exclusive 的值为 True)。对于 SQL 数据库服务器上的数据库表，当以互斥方式被一个用户打开时，其他用户可以读取该表中的数据，但不能修改表中的数据。有些数据库服务器不支持这种方式，这要具体参看有关的数据库服务器的文档。

10.ReadOnly 属性和 CanModify 属性

ReadOnly 和 CanModify 这两个属性都是布尔型属性。ReadOnly 属性决定用户是否能够对表中的数据进行读写。ReadOnly 为 True 时，用户只能读取表中的数据；ReadOnly 为 False 时，用户可以读写表中的数据(假设数据库已授权用户能够读写其中的数据库表)。CanModify 属性是一个只读属性，用户不能够修改其属性值。CanModify 属性反映了用户对数据库表拥有的实际特权。当 ReadOnly 为 True 时，CanModify 将自动的被置为 False；当 ReadOnly 为 False 时，如果数据库允许用户对表进行读写时，CanModify 为 True，否则

CanModify 为 False。当 CanModify 为 False 时，数据库表是只读的，不能将其置成编辑状态或插入状态；当 CanModify 属性为 True 时，虽然数据库表对应的数据集部件可以置成编辑和插入状态，但是这并不意味着用户能够插入和修改表中的数据，因为这还要受到其他因素的限制，如用户对 SQL 数据库服务器的访问权限等的限制。

TTable 部件还有其他一些属性请参看联机帮助，这里介绍的属性都是最常用的。

2.2.1.2 TTable 部件的方法及应用

1.设定数据库表的使用范围的方法

在实际应用中的数据库表中常常存放着大量的数据信息，其中包含着很多的记录，而常常应用程序可能只需对其中一部分记录进行操作。因此，为应用程序指定一个使用范围就显得特别重要了。为方便有效的指定数据库表的使用范围，Delphi 为 TTable 部件提供了下列方法以供使用：

● SetRangeStart、EditRangeStart、SetRangeEnd 和 EditRangeEnd 方法

SetRangeStart 方法用于指定检索范围的起始记录。调用 SetRangeStart 方法之后，可以为起始记录的一个或多个字段指定相应的字段值。SetRangeEnd 方法用于指定检索范围的结束记录。调用 SetRangeEnd 方法之后，可以为结束记录的一个或多个字段指定相应的字段值。其它的两种方法使用范围基本一致。

● SetRange([Start Values],[End Values])方法

SetRange 方法包含了 SetRangeStart 和 SetRangeEnd 方法的功能。它可以同时指定检索范围的起始和结束记录。起始记录和结束记录的字段值以数组形式送给 SetRange，其基本形式是：

SetRange([起始值]，[结束值])

● ApplyRange 方法

根据 SetRangeStart、SetRangeEnd 或 SetRange 方法确定检索的起始和结束记录，Delphi 将具体设定一个检索范围。调用 ApplyRange 方法之后，应用程序只能对检索范围内的记录进行有关的操作。

● CancelRange 方法

CancelRange 方法的作用与 ApplyRange 方法的作用是相反的。CancelRange 方法可以取消为表设定的检索范围。调用 CancelRange 方法之后，数据库应用程序可以对表中全部记录进行有关的操作。

值得注意的是：如果使用的是 paradox 表或 dBASE 表，在调用 SetRangeStart、SetRangeEnd 以及 SetRange 方法时，只能为表中的索引字段或定义的索引指定相应的字段值，以设定检索范围。如果使用 SQL 数据库服务器中的数据库表，可以为 IndexFieldNames 属性中指定的字段指定相应的字段值。

这里有一个例子，例如假设部件 Table1 与 Customer.DB 表相连。Customer.DB 中一个索引字段是 CustNo，同时数据库应用程序窗体中有两个编辑框 StartVal 和 EndVal 用于输入起始、结束记录的字段 CustNo 的值。下面的程序代码便可以设定一个检索的范围：

```
Tabel1.SetRangeStart;      {指定检索范围的起始记录}
Tabel1CustNo.AsString:= StartVal.Text {为起始记录的 CustNo 字段指定字段值}
Tabel1.SetRangeEnd;        {指定检索范围的结束记录}
if EndVal.Text <> '' then
Tabel1CustNo.AsString := EndVal.Text; {为结束记录的 CustNo 字段指定字段值}
Tabel1.ApplyRange;         {根据检索范围的起始、结束记录设定检索范围}
```

上面的程序代码中，当为结束记录的 CustNo 字段指定字段值时，首先要检查 EndVal 的值是否为空。如果 EndVal 的值为空，那么设定的检索范围没有包含一条记录，因为没有任何记录的字段值小于 NIL；如果 StartVal 的值为空，那么检索范围将从表中的第一条记录开始，因为表中任何记录的字段值都大于空(NIL)。

上述程序代码的功能可以同样用 SetRange 方法实现，程序代码如下：

```
If EndVal.Text <>'' then
Tabel1.SetRane([StartVal.Text].[EndVal.Text]);
Table1.ApplyRange;
```

EditRangeStart 和 EditRangeEnd 方法的使用完全类似于 SetRangeStart 和 SetRangeEnd 方法，只是调这两个方法是设定一个可编辑的范围。

又如：假设一个表中的一个索引包含两个字段 LastName 和 FirstName，为索引中的一个字段或多个字段指定相应的字段值，设定数据库表的使用范围。

```
Table1.SetRangeStart;
Table1.FieldByName('LastName').Asstring := 'Smith';
Table1.SetRangeEnd;
Tabel1.ApplyRange;
```

上述代码设定的范围包括 LastName 字段的值大于或等于 Smith 的所有记录。而下面的代码设定的范围则包括 LastName 字段的值大于或等于 Smith，且 FirstName 字段的值大于或等于'J'的记录。

```
Table1.SetRangeStart;
Table1.FieldByName('LastName').Asstring := 'Smith';
Table1.FieldByName('FirstName').Asstring := 'J';
Table1.SetRangeEnd;
Tabel1.ApplyRange;
```

2.查找数据库表中记录的方法

如果想查找数据库表中的记录，必须想指定查找记录的一些字段的字段值，然后在表

中进行检索，检索出与查找值相匹配的记录来。如果数据库应用程序是在 Paradox 或 dBASE 数据库中的表中查找记录，那么查找值所对应的字段必须是表中的关键字段或辅助索引字段。如果查找 SQL 数据库服务器中的表，那么查找值必须是表的 IndexFieldNames 属性中指定的字段。

Delphi 提供了两种方式在数据库表中查找记录：Goto 方式和 Find 方式。这两种方式十分相似，它们的主要区别在于为查找指定查找值的方法不一样。

● 使用 Goto 方式

使用 Goto 方式进行数据查找使用的方法有 SetKey 方法、GotoKey 方法和 GotoNearest 方法。其实际步骤如下：

(1)确保要查找的字段是关键字段或辅助索引字段；

(2)调用 SetKey 方法把与表对应的 TTable 部件置成查找状态；

(3)把查找值赋给相应的字段；

(4)调用 GotoKey 方法，并测试它的返回值检验查找是否成功。

假设 Table1 对应的表中第一个字段是关键字段，Edit1 是数据库应用程序窗体中的一个编辑框，用户可以通过 Edit1 输入查找值。下面的代码将通过 Goto 方式进行查找。

```
Table1.SetKey;                          {将 Table1 置成查找状态}
Table1.Field[0].AsString := Edit1.Text; {指定查找值}
Table1.GotoKey;                         {进行查找}
```

在上面的程序代码中的最后一行是根据用户指定的查找值，在表中执行查找。查找的结果有两种，也许成功也许失败，这是由调用 GotoKey 方法之后返回的布尔值来决定。如果返回 True，那么说明查找成功，并且记录指针会指向与查找值匹配的记录；如果返回 Fale，那么说明查找失败，记录指针的位置不发生变化。下面的代码可以测试调用 GotoKey 方法之后的返回值，告知用户查找是否成功，如下：

```
Table1.SetKey;
Table1.Field[0].AsString:= 'Smith';
If not Table1.GotoKey then
ShowMessage('记录没找到')
```

在这一段代码中，如果在表中没有找到第一个字段值为 Smith 的记录，该应用程序会弹出一个对话框告知用户“记录没有找到”。

如果在表中存在多个关键字段或辅助索引中包含多个字段时，在进行查找时只想为第一个字段指定查找值，那么必须要设置 TTable 部件的 KeyFieldCount 的属性值为 1。如果想为多个字段指定查找值，只能为相邻的字段指定查找值。例如辅助索引中共有三个字段，那么只能为第一个字段、第一和第二个字段、第一和第二以及第三个字段指定查找值，而不能为第一和第三个字段指定查找值。

GotoNearest 方法的使用与 GotoKey 方法完全一样，只是它用于不精确查找。它不要