李维作品系列

Delphi 7
高效数据库程序设计

李维 著

Delphi 6/Kylix 1/2/3 适用

Borland新一代dbExpress提供了跨平台的高效率的数据引擎。dbExpress不但提供广泛的关系型数据库存取能力，并且适用于客户机/服务器、Web和多层的应用系统。本书不但深入讨论dbExpress的使用技巧，更辅以本章的范例让读者完全掌握Delphi/Kylix的dbExpress和DataSnap技术。

附赠

机械工业出版社
China Machine Press

第一部分

dbExpress基本功能篇

第1章 dbExpress组件、概念、技术和应用程序

Delphi 6/7的重要功能之一便是推出了新一代跨平台的数据访问引擎——dbExpress。dbExpress是一组新的组件、技术和驱动程序，以允许程序员使用它连接到各种数据源，再配合不同的数据库连接 DLL文件，让程序员可以处理后端数据库中的数据。由于dbExpress具备了跨平台的能力，能够同时在 Windows和Linux平台以及未来的.NET上使用，更提供了快速的数据处理能力，让程序员能够开发出更有效率的数据库应用程序，因此势必成为以后 C++Builder/Delphi和Kylix的核心数据访问技术。

这些新的dbExpress组件除了新进入Delphi世界的程序员需要学习之外，即使是使用Delphi已经有一段时间的有经验的 Delphi程序员也需要上手的时间。此外，dbExpress提供了强大的功能，允许 Delphi程序员微调它的执行行为，并且访问低层的核心信息。同时dbExpress也能够与DataSnap技术（这是Delphi 6/7中的名称，在旧的Delphi版本中称为MIDAS）结合以便让程序员能够同时开发单机、 Briefcase、主从结构和瘦客户类型的数据库应用程序，让程序员能够使用一组组件和技术同时开发数种类型的应用系统。

本书将会讨论dbExpress的所有强大功能，不过万丈高楼平地起，本章将从说明如何使用这些新的dbExpress组件开始，一步一步地带领各位学习到 dbExpress最精髓的核心技术。

1.1 dbExpress组件

在Delphi 7中，dbExpress组件组包含了7个组件，这些组件的功能就是让应用程序连接后端数据库，访问数据表中的数据，把修改的数据更新回数据库中以及让程序员观察dbExpress向后端数据库下达的命令等。简单地说，这些组件涵盖了数据库应用程序所有必要的功能。图 1-1是dbExpress组件面板中的所有 dbExpress组件。

图1-1 Delphi 7的dbExpress组件组

下面的表格概要地说明了每一个 dbExpress组件的基本功能，在稍后的章节中将会详细讨论如何使用每一个 dbExpress组件。

组件名称	功 能
TSQLConnection	与后端数据库建立连接的组件
TSQLDataSet	可以用来执行SQL语句、执行存储过程或是直接连接数据库中特定数据表的组件。它算是一个通用的组件。TSQLDataSet同时具备了类似于TSQLQuery、TSQLStoredProc和TSQLTable组件的能力
TSQLQuery	用来执行SQL语句的组件
TSQLStoredProc	用来执行数据库中的存储过程（Stored Procedure）的组件
TSQLTable	用来连接数据库中特定的数据表的组件，类似于BDE中的TTable组件
TSQLMonitor	可以观看和检视客户端向后端数据源发出的SQL语句的组件。程序员可以使用它来调试或是调整应用程序的性能
TSimpleDataSet	允许dbExpress修改数据的组件，可以结合Delphi的数据感知组件以访问数据

在Delphi 7.0和Kylix 3.0中，dbExpress目前正式支持6种数据库，这些支持的数据库以及对应的dbExpress驱动程序总结在下面的表格中。

数据库名称	dbExpress驱动程序
InterBase 6.5/6.x	DBEXPINT.DLL
DB2 7.2	DBEXPDB2.DLL
Oracle 9i	DBEXPORA.DLL
MySQL 3.23.49	DBEXPMYS.DLL
MS SQL Server 2000	DBEXPMSS.DLL
Informix SE	DBEXPINF.DLL

虽然目前dbExpress只支持6种数据库，但是Borland已经声明将继续提供其他数据库的dbExpress驱动程序，例如Sybase System 12等。在新的dbExpress驱动程序开发完毕之后，Borland会公布在网站上让程序员下载，并且包含在随后版本的Delphi和Kylix中。

学习dbExpress组件最好的方法就是直接使用它来开发数据库应用程序，在下一个小节中，我们将立刻开始学习dbExpress组件组，让程序员快速地学习如何使用这些新的组件开发应用系统。

1.2 建立第一个dbExpress数据库应用程序

现在就让我们快速地使用dbExpress来开发一个数据库应用程序，学习如何使用dbExpress组件来访问数据。要连接数据库并且从其中访问数据，程序员可以使用下面的三个步骤来完成这个工作：

1) 使用TSQLConnection组件连接数据库。
2) 使用TSQLDataSet组件访问数据。
3) 在数据感知组件中显示数据。

现在就让我们一步一步地完成上面的三个步骤。

步骤1：使用TSQLConnection组件连接数据库

首先点击Delphi的File|New|Application菜单建立一个新的Delphi应用程序，接着点击组件面板中的dbExpress选项卡，选择第一个组件TSQLConnection并且将它放入到应用程序的主窗体，如图1-2所示。有了TSQLConnection组件之后，现在我们需要让它连接到数据库服务器，在这个范例中是使用 InterBase，读者可以使用其他数据库，例如Oracle或MySQL等。

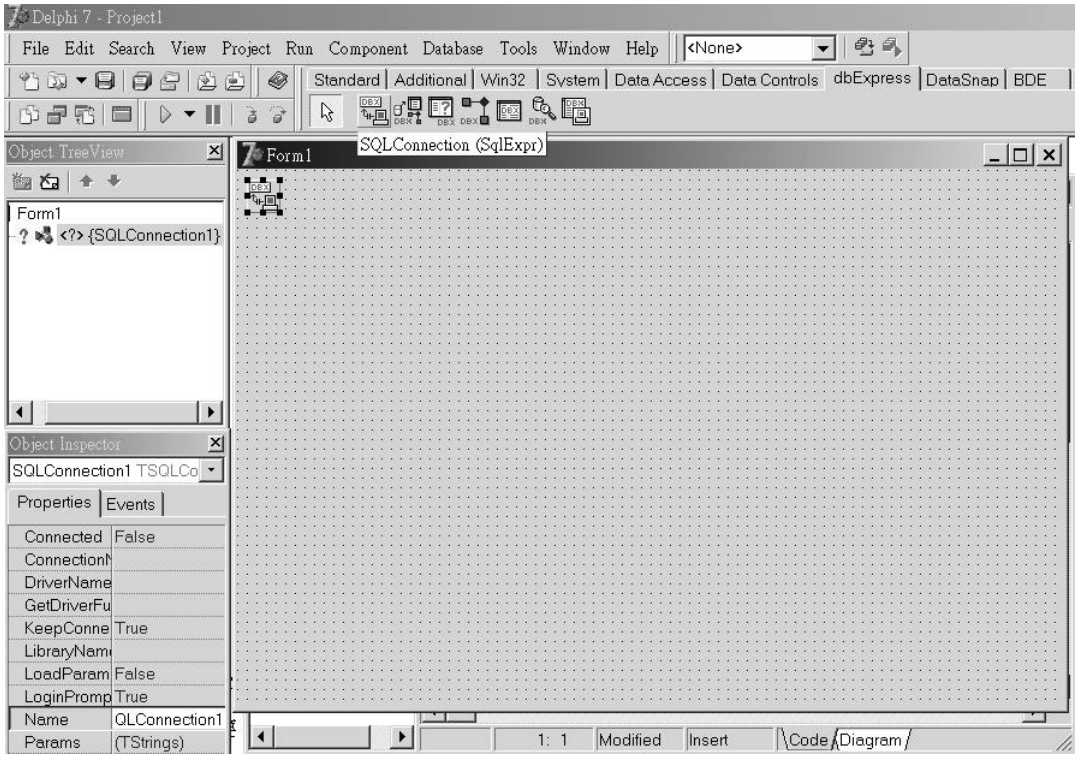


图1-2 在主窗体中放入TSQLConnection组件以连接数据库

要使用TSQLConnection连接数据库，请使用鼠标双击 TSQLConnection，此时会出现TSQLConnection的组件编辑器，如图1-3所示。在图1-3中列出了目前dbExpress内置的连接或是用户新增的连接，由于现在本范例要使用 InterBase作为连接的数据库，因此请使用鼠标点击上方的“+”按钮以建立一个新的连接。

此时Delphi会显示图1-4所示的新连接对话框，请在这个对话框中选择使用InterBase驱动程序，并且输入一个连接名称。在这个范例中使用了 CHINESEDEMO作为本范例的连接名称。

接着Delphi会显示图1-5所示的对话框，要求输入CHINESEDEMO真正的数据库

路径和名称信息，请按照图1-5那样输入数据库的实际位置。由于CHINESEDEMO使用的InterBase数据库是使用BIG5内码建立的，因此请读者记得将ServerCharSet的特性值改为BIG5。在输入了数据库实际位置之后，读者可以点击对话框上方的“˘”按钮，以测试是否可以正确地连接到数据库。最后确定一切正确之后，点击OK按钮以完成设置TSQLConnection组件的步骤。

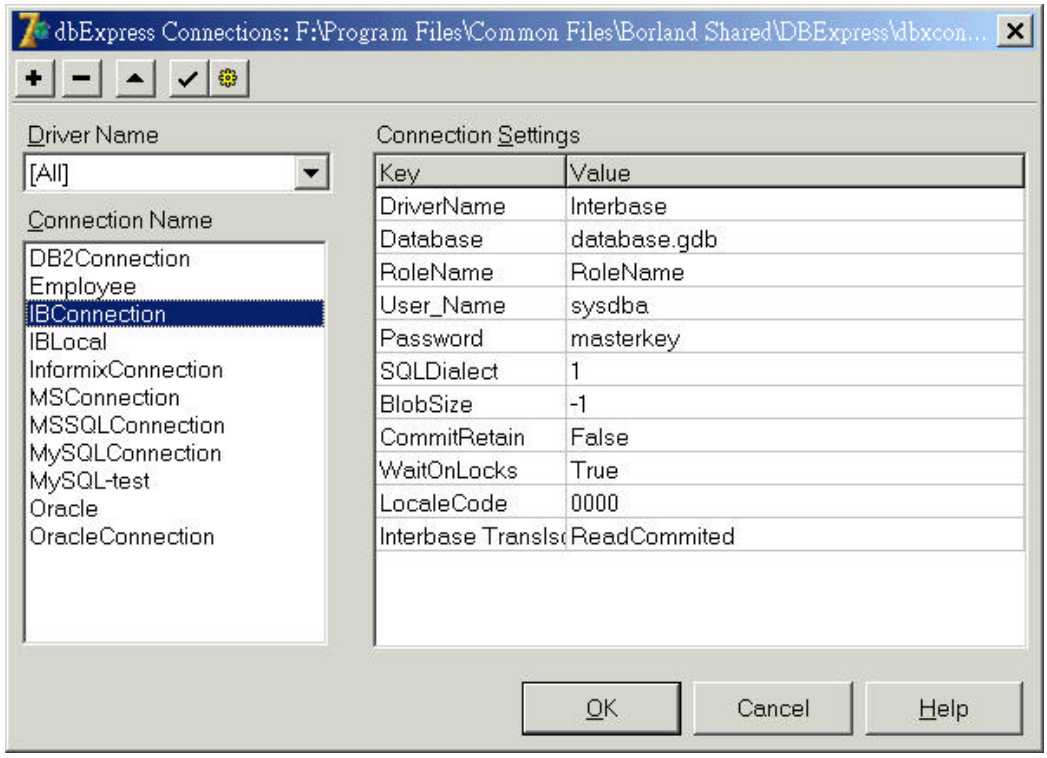


图1-3 TSQLConnection组件的组件编辑器

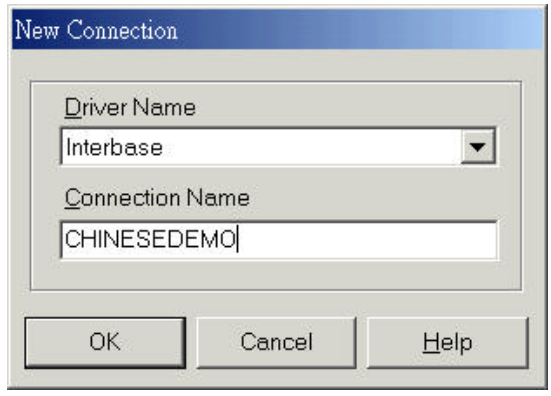


图1-4 dbExpress的新数据库连接对话框

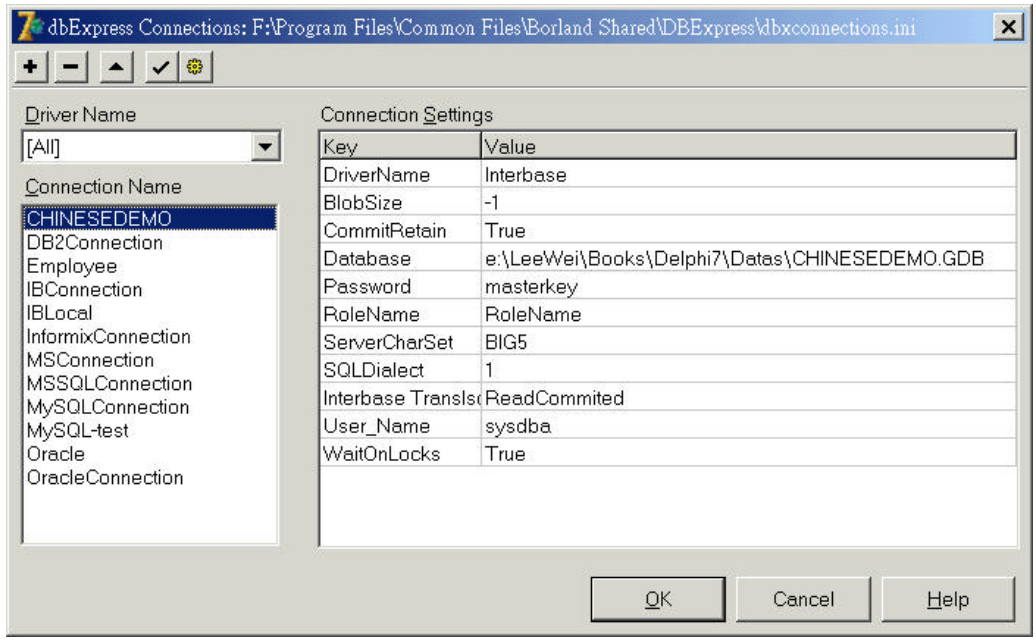


图1-5 输入CHINESEDEMO要连接的InterBase数据库

读者可以在本书的光盘中找到CHINESDEMO.GDB这个InterBase数据库。

如果读者仔细观察图1-5所示的对话框，便会发现刚才设置的CHINESEDEMO这个InterBase连接信息事实上是存储在\Borland Shared\DBEXpress目录下的dbxconnections.ini文件中，这是一个文本文件，读者也可以使用文字编辑器，例如NotePad或是Delphi的编辑器来修改其中的内容。

现在请点击对象检视器，设置TSQLConnection的LoginPrompt特性值为False，以避免出现登录对话框，最后再把Connected特性值设置为True。如此一来我们就成功地连接到InterBase服务器了（当然，读者的InterBase必须正在执行）。

步骤2：使用TSQLDataSet组件访问数据

现在再从dbExpress选项卡中选择第二个组件TSQLDataSet，并且将它放到主窗体中。先在对象检视器中将它的SQLConnection特性值设置为步骤1放入的SQLConnection1，再点击它的CommandText特性值。此时Delphi便会显示CommandText特性值的特性值编辑器，让程序员可以使用可视化的方式下达SQL命令。图1-6便是激活CommandText特性值编辑器的画面。

当CommandText特性值编辑器出现时，它会自动地从连接的数据库中取得所有目前可以被看到的数据表名称，放入到左上方的Tables列表框中，而把选择的数据表

的字段信息呈现在左下方的列表框中，真正的SQL命令则在右边的Memo控件中。至于什么数据表会出现在左上方的 Tables列表框中则是由 TSQLConnection的TableScope特性值来决定的。

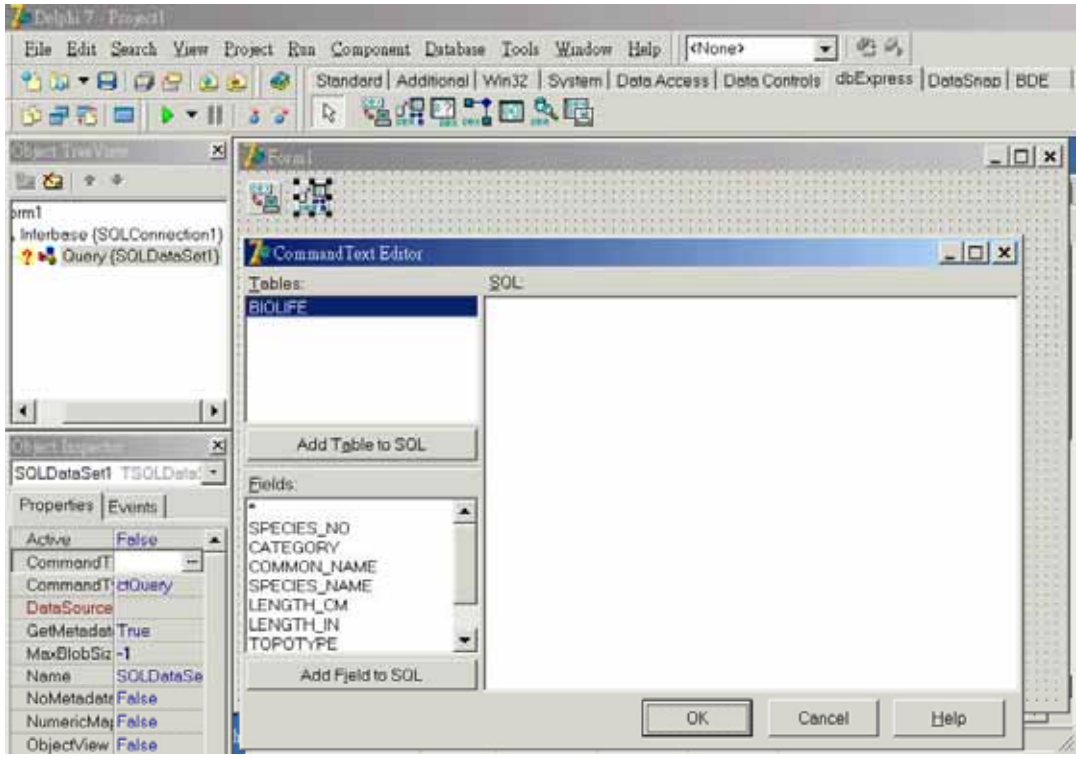


图1-6 TSQLDataSet的CommandText特性值编辑器

由于CHINESEDEMO中只有一个数据表BIOLIFE，因此请使用鼠标点击左上方的BIOLIFE，再点击左下方的“*”以代表要从BIOLIFE数据表中取得所有字段的数据，如此一来CommandText特性值编辑器便会在右边的SQL窗口中自动产生Select * from BIOLIFE的SQL命令，如图1-7所示。

请点击OK按钮，Delphi便会把刚才完成的SQL命令存储在TSQLDataSet组件的CommandText特性值中。接着请在主窗体中放入一个 TDataSource组件，将它的DataSet特性值设置为刚才的TSQLDataSet组件，再放入一个TDBNavigator组件，将它的DataSource特性值设置为刚放入的TDataSource组件。

现在我们就可以通过把 TSQLDataSet的Active特性值设置为 True将数据从BIOLIFE数据表取到应用程序中。接着让我们激活TSQLDataSet的字段编辑器，把代表BIOLIFE每一个字段的字段对象加入应用程序中。把字段对象加入应用程序中的好处是程序员可以使用Delphi的对象检视器来设置字段对象的特性，例如字段显示的中文名称，或是设置字段显示的栏宽等。

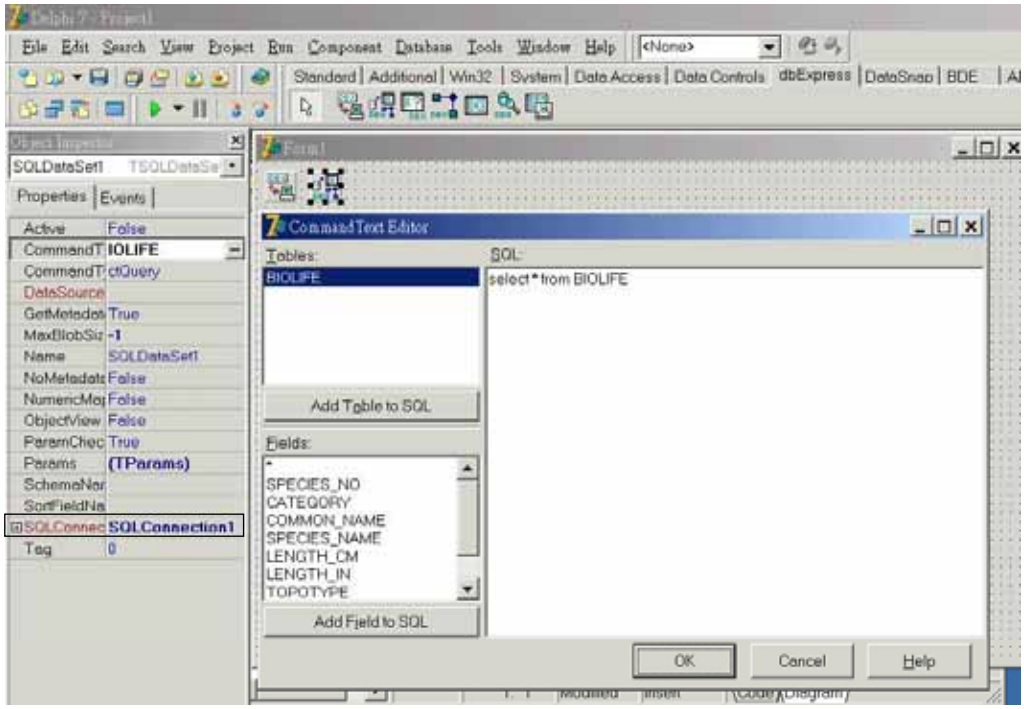


图1-7 使用CommandText特性值编辑器完成SQL命令

请点击主窗体中的TSQLDataSet组件，按下鼠标右键，Delphi会显示一个快捷菜单，菜单中的第一个选项Fields Editor...便可以激活字段编辑器，如图1-8所示。

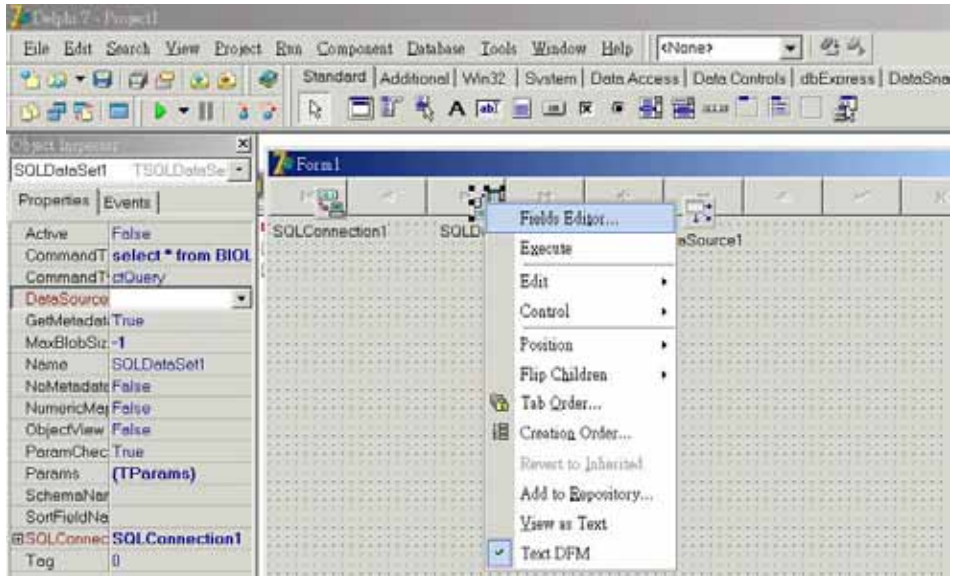


图1-8 激活TSQLDataSet的字段编辑器

请点击这个选项，Delphi便会显示一个空白窗口，请在这个空白窗口中再点击鼠标右键，Delphi便会显示一个快捷菜单，点击这个菜单中的Add all fields选项，以便将代表BIOLIFE数据表中每一个字段的字段对象加入应用程序中，如图 1-9所示。接着Delphi便会把所有的字段对象加入刚才的空白窗口中，现在我们就可以点击这个窗口中的每一个字段对象，然后使用对象检视器来设置它们的特性值。

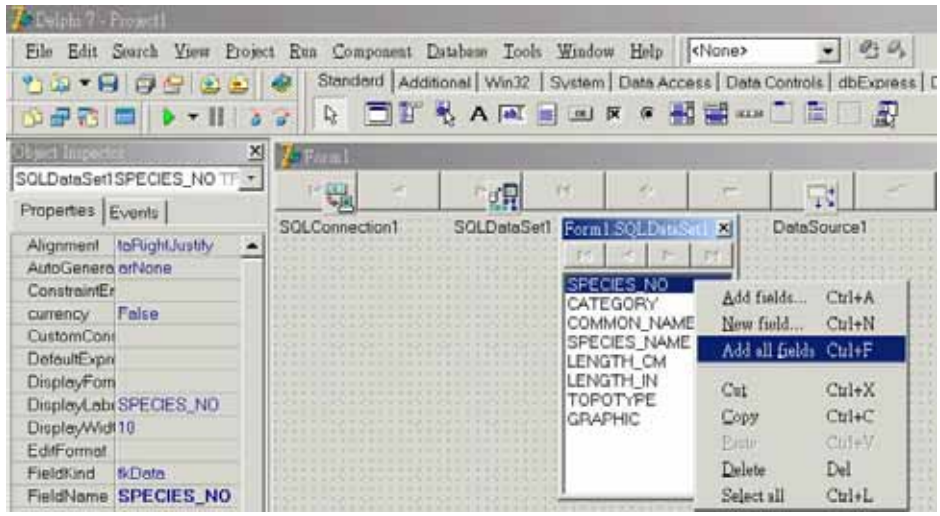


图1-9 加入所有的字段对象

现在我们希望应用程序在执行时显示中文的字段名称，而不是数据表中字段的英文名称，那么我们可以点击图 1-9中的每一个字段对象，然后在对象检视器中设置字段对象的DisplayLabel特性，接着输入这个字段的中文名称即可。例如图 1-10便是设置SPECIES_NO这个字段的中文名称的画面。

在我们一一设置完每一个字段对象的 DisplayLabel特性值之后，便可以选择所有的字段对象，然后把它们拖曳到主窗体中，那么 Delphi便会自动地在主窗体中产生能够适当显示每一个字段对象的数据感知组件，并且在这些数据感知组件中显示数据。例如图1-11便是把图1-10中的所有字段对象拖曳到主窗体中后的画面。

现在我们已经完成了第一个使用 dbExpress组件实现的范例数据库应用程序，请执行它，此时我们便可以看到类似图 1-12的画面。dbExpress组件果然可以顺利地从InterBase中访问数据，并且显示在数据感知组件中，就和 Delphi原本的BDE/IDAPI组件一样方便。

但是请读者仔细观察图 1-12的画面，读者会发现图 1-12中的TDBNavigator中所有与修改数据有关的按钮都被暂停使用，例如代表修改数据的“▲”按钮和新增数据的“+”按钮。这意味我们无法使用这个范例应用程序来修改 BIOLIFE数据表中的数据，也代表由dbExpress组件开发的数据库应用程序是无法修改的。

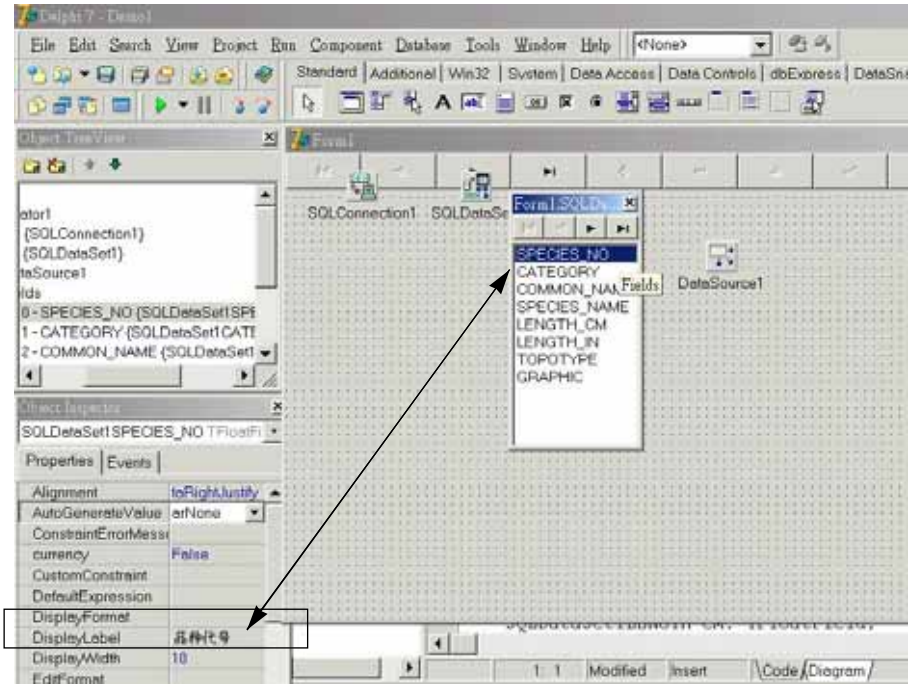


图1-10 使用对象检视器设置字段对象的特性值

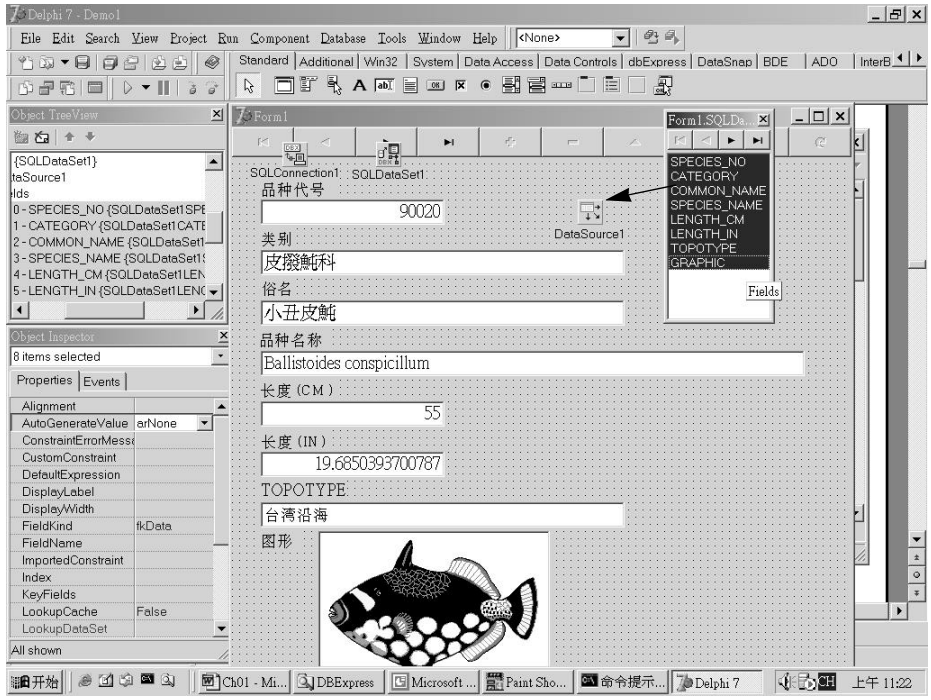


图1-11 将所有字段对象拖曳到主窗体中

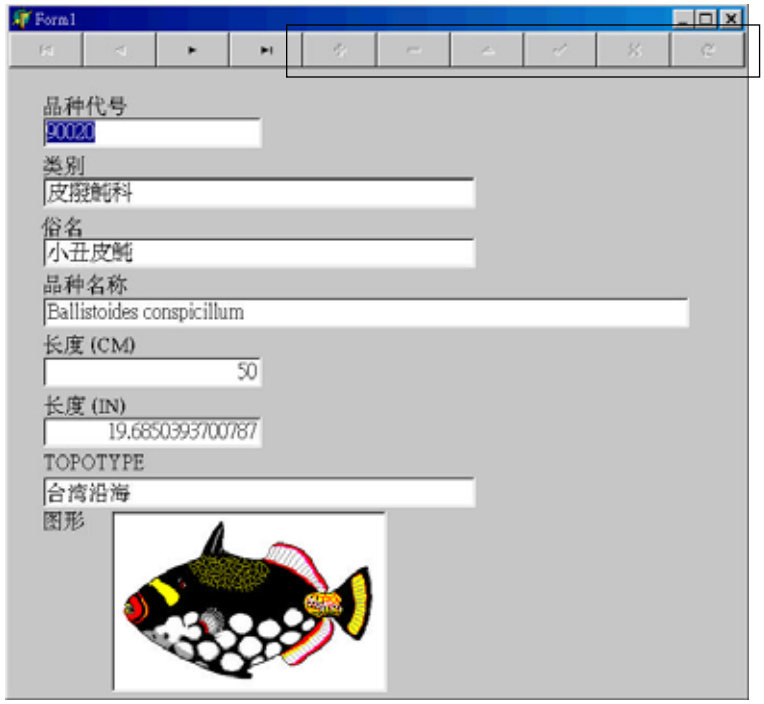


图1-12 范例应用程序执行的画面，请注意 TDBNavigator中所有与修改数据有关的按钮都被禁用，代表应用程序无法修改数据

现在如果读者点击向下一个记录的按钮移动到随后的记录，接着再点击往前的按钮欲把目前的记录移动到前一个记录时，此时范例应用程序便会显示下面的错误消息。

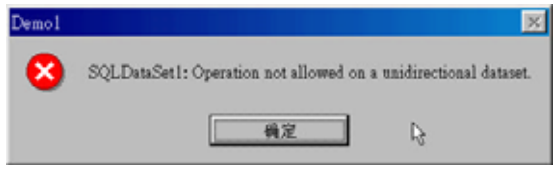


图1-13 点击TDBNavigator中的前一个记录的按钮时，范例应用程序会发生错误

这个错误消息的意思是由 dbExpress组件开发的应用程序使用只能向后走的单向游标，而无法向前走回到前面的记录。

也许读者会觉得很奇怪，既然 dbExpress无法修改数据，也无法任意移动目前记录的位置，那么dbExpress不是一点用处都没有吗？如何能够用它来开发实际的数据库应用程序呢？

当然不，这是因为 dbExpress是使用与以往 BDE组件不一样的方式来设计的，因此如果读者按照使用 BDE的概念来使用dbExpress便会觉得奇怪。事实上，dbExpress不但能够做到和 BDE组件一样的功能，甚至还有比 BDE更多的功能，在性能上也胜

过BDE。在本书进一步说明如何使用dbExpress修改数据之前，先说明dbExpress组件的一些重要的概念以及它的设计结构，如此一来读者将会更清楚地了解dbExpress的设计原理，在稍后的章节中也将更能掌握dbExpress。

1.3 使用dbExpress的概念

dbExpress组件组中与访问数据有关的组件，例如TSQLDataSet、TSQLTable等都是从TDataSet类继承下来的，因此它们提供了所有和TDataSet一样的功能，也可以和Delphi的数据感知组件使用在一起以开发数据库应用程序。图1-14显示了这些dbExpress组件的类结构图。

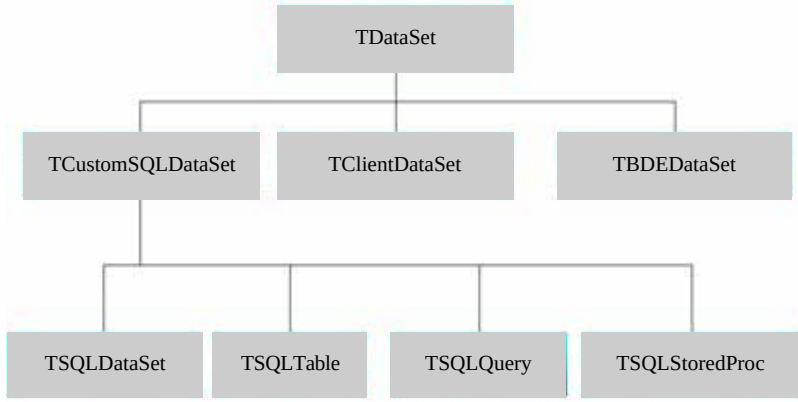


图1-14 dbExpress的类结构图

虽然TSQLDataSet等组件是从TDataSet继承下来的，但是它们与其他也从TDataSet继承下来的组件（例如TBDEDataSet等组件）不一样的地方是，通过这些dbExpress组件取得的结果数据集是只读的。这也就是说由这些组件取得的数据是不能修改的，而且由这些dbExpress组件取得的结果数据集的数据访问方式是只能由前往后访问，而无法由后往前访问。

如果程序员想能够修改由这些dbExpress取得的数据，或是想在结果数据集中以任意的方式移动目前记录的位置，那么程序员可以使用两种方法来达成：

1) 在应用程序中再搭配使用组件面板中Data Access选项卡中的TDataSetProvider和TClientDataSet组件。

2) 使用dbExpress组件面板中的TSimpleDataSet组件。

本书稍后将会同时说明如何使用这两种方式，现在先让我们说明如何使用同样位于组件面板dbExpress选项卡中的TSimpleDataSet组件。

TSimpleDataSet组件是Delphi 7用来帮助程序员通过dbExpress组件修改数据的组件。基本上，使用TSimpleDataSet组件等同于使用TSQLDataSet组件加

TDataSetProvider组件，再加上TClientDataSet组件。TSimpleDataSet不但能够访问数据，更能够让程序员修改数据和移动目前记录的位置。

基本上，TSimpleDataSet组件是使用TSQLDataSet组件从后端数据源中取得数据，再通过内部的缓存（cache）机制管理数据，通过内部缓冲（buffering）机制允许程序员任意移动目前记录位置，并且在程序员需要将修改的数据写回数据源时，根据它内部维护的信息自动地帮程序员产生修改数据的SQL语句，再通过TSQLDataSet组件使用这些自动产生的SQL语句把修改的数据更新回数据源。因此TSimpleDataSet组件等于是帮程序员编写管理数据以及把修改的数据更新回数据源的程序代码，而无需程序员自己编写这些程序代码。

图1-15就是TSimpleDataSet组件的类结构图，从图中可以看到TSimpleDataSet是从包含缓存机制的TCustomClientDataSet组件继承下来的。

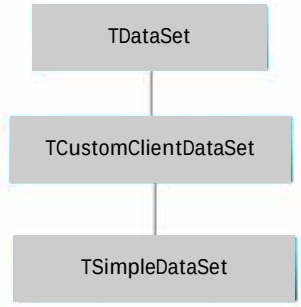


图1-15 TSimpleDataSet的类结构图

当程序员使用TSimpleDataSet时就不需要再搭配使用TSQLConnection和TSQLDataSet，因为TSimpleDataSet自己会在内部建立暂时的TSQLConnection组件、TDataSetProvider组件以及一个TInternalSQLDataSet组件。TSimpleDataSet会使用内部建立的TSQLConnection连接至数据库，再使用内部建立的TInternalSQLDataSet和TDataSetProvider处理数据。由于TSimpleDataSet内部使用了TDataSetProvider组件，因此提供了修改数据的能力，比使用TSQLDataSet只能提供数据查询的能力方便多了。

此外，虽然程序员在Delphi 7中是使用dbExpress组件访问和处理数据，但是在dbExpress组件之下，程序员仍然必须拥有每一个特定数据库的dbExpress驱动程序才可以访问特定的数据库。当程序员像图1-5那样使用TSQLConnection组件指定连接特定的数据库时，dbExpress便会加载与此数据库相关的dbExpress驱动程序以访问这个数据库。

但是加载的dbExpress驱动程序仍然需要调用特定数据库的客户端驱动程序，才能够真正访问特定数据库。例如，要使用dbExpress访问Oracle，那么除了需要dbExpress中用于访问Oracle的DBEXPORA.DLL驱动程序之外，由于DBEXPORA.DLL直接调用Oracle的Oracle Call Interface，因此还需要Oracle客户端的OCI.DLL，否则dbExpress仍然无法访问Oracle数据库。

例如，图1-16就是dbExpress在访问数据库时实际的结构。当dbExpress组件要访问Oracle时，就使用刚才描述的流程。例如dbExpress要访问InterBase时，就需要加载dbExpress中用于访问InterBase的DBEXPINT.DLL驱动程序以及InterBase客户端的引擎GDS32.DLL。

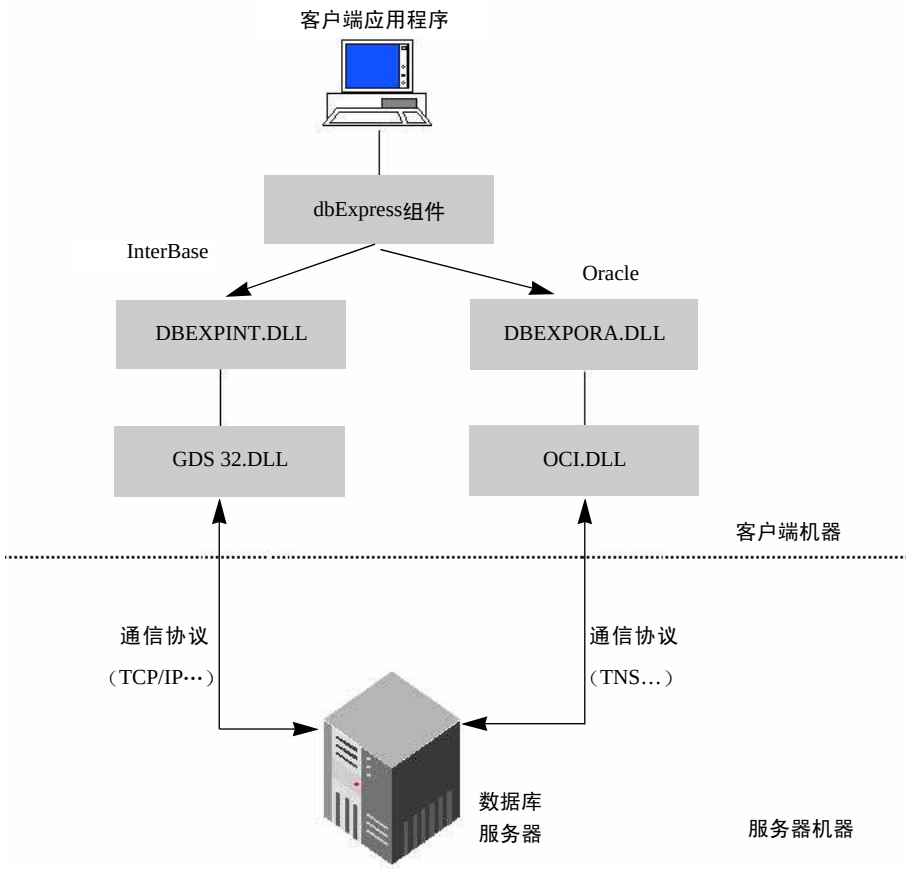


图1-16 TSQLClientDataSet的类结构图

在dbExpress组件通过驱动程序取得了结果数据集之后，程序员可以再使用TSimpleDataSet来修改结果数据集中的数据，因为TSimpleDataSet包含了内部缓存的机制，可以自动帮助程序员完成这些工作。如果我们更详细地说明什么是TSimpleDataSet内部的缓存机制，那么答案就更清楚了，这个缓存机制事实上就是以往Delphi中的MIDAS技术。Delphi 7中的MIDAS已经是第5个版本了（Delphi 6中的MIDAS是第4个版本），它在Delphi 6中被重新命名为DataSnap。Delphi 7中的DataSnap已经从以往的分布式应用系统结构技术转换为跨平台的标准数据访问技术。这个意思是说，MIDAS 5不但可以用来开发分布式应用系统，在Delphi 7中也成为开发主从结构、数据库和单机应用程序的标准数据访问技术，并且能够同时在

Windows平台的Delphi 7中以及Linux平台的Kylix中使用。因此DataSnap已经成为Delphi的核心技术之一，所有的Delphi程序员都必须了解并且纯熟地掌握DataSnap技术，当然，本书也会充分地说明如何善用DataSnap技术。

因此在Delphi 7中开发数据库应用程序的结构就如图1-17所示的一样，应用程序通过dbExpress访问数据，再由DataSnap（MIDAS 5）来管理和修改数据。

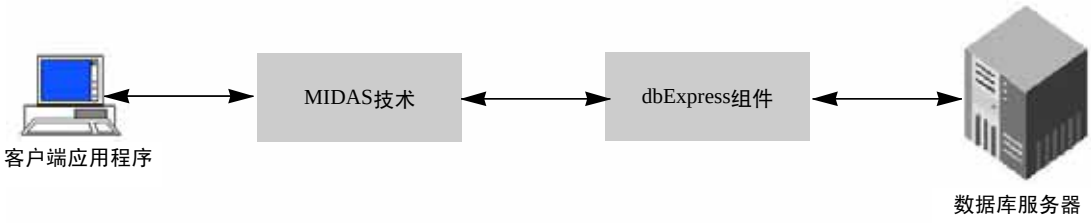


图1-17 TSQLClientDataSet的类结构图

图1-18是更详细的结构图，Delphi 7应用程序通过dbExpress组件访问数据，而dbExpress使用SQL语句从数据源中取得数据，当数据源根据SQL语句产生结果数据集并且返回给dbExpress组件之后，dbExpress组件会再把结果数据集交由DataSnap技术管理，以便程序员能够对结果数据集中的数据进行处理和修改。

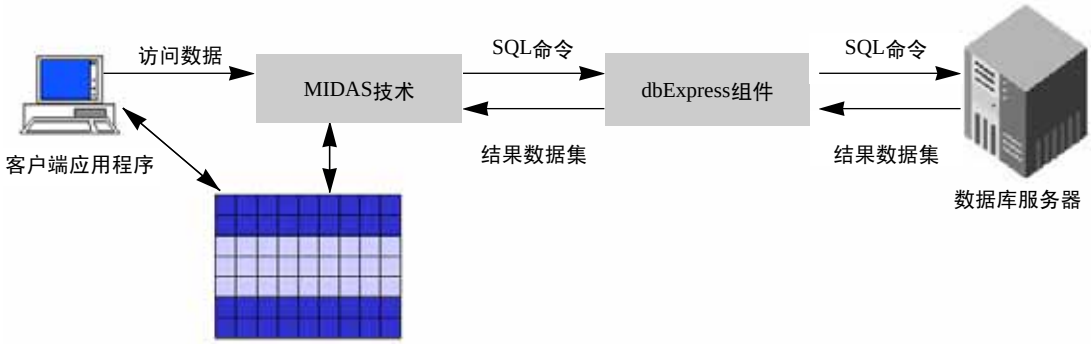


图1-18 TSQLClientDataSet的类结构图

从图1-18中我们也可以了解到，通过TSimpleDataSet取得的数据事实上就是由DataSnap在缓存内存中维护的数据。几乎所有通过TSimpleDataSet的方法或特性值处理的数据都是存储在这些缓存内存中的数据，只有当程序员真的需要把缓存内存中的数据更新回数据源时，DataSnap才会通过SQL语句把经过修改的数据更新回数据库，在稍后的小节中会说明如何修改数据并且把数据真正更新回数据源中。

1.4 使用dbExpress修改数据

现在本书已经说明了dbExpress的结构和原理，在本书随后的章节中会逐步详细说明dbExpress技术的精髓，让程序员能够精确地掌握dbExpress技术。不过在学习

dbExpress的其他功能之前，先让我们以实际的范例来说明如何使用 dbExpress任意地移动记录位置和修改数据。

在本小节中也将同时说明如何使用 TSimpleDataSet以及使用TSQLDataSet搭配TDataSetProvider和TClientDataSet组件，以便证明在前面小节中提到的修改数据的方式。首先让我们延续前面 1.2小节的范例来说明如何完整地处理 CHINESEDEMO的BIOLIFE数据表中的数据。第一步是让 1.2小节中使用的范例程序搭配 TDataSetProvider和TClientDataSet组件。

1.4.1 使用TSQLDataSet搭配TDataSetProvider和TClientDataSet组件

请回到Delphi 7集成开发环境，开启范例应用程序的主窗体，然后在主窗体中加入组件面板Data Access选项卡中的TDataSetProvider和TClientDataSet组件，最后再加入dbExpress选项卡中的TSQLDataSet组件，设置TSQLDataSet组件的SQLConnection特性值为窗体中的SQLConnection1，再设置它的SQL特性值为Select * from BIOLIFE，如图1-19所示。

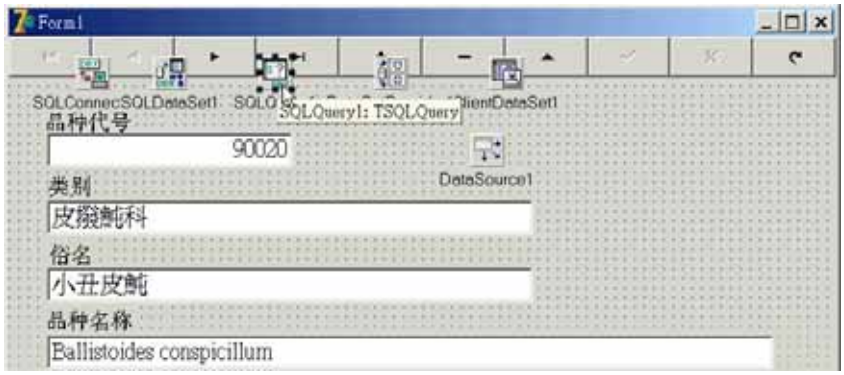


图1-19 在范例主窗体中加入TDataSetProvider以及TClientDataSet组件

在前一小节中，本书说明了由 dbExpress取得的结果数据集虽然无法修改，但是只要搭配Delphi的DataSnap技术就能够允许应用程序修改其中的数据。因此现在我们要做的就是在范例应用程序中加入 DataSnap的功能。

在加入了TDataSetProvider组件之后，使用对象检视器设置它的 DataSet特性值为主窗体中原先的TSQLDataSet组件，如图1-20所示。

TDataSetProvider组件能够把属于 TDataSet类的组件（例如 TSQLDataSet，请参考图1-14的类结构图）输出给TClientDataSet组件，以便让TClientDataSet组件能够管理输出的TDataSet组件中的结果数据集数据。

接着使用对象检视器设置刚才加入的 TClientDataSet组件的ProviderName特性值为主窗体中的TDataSetProvider组件，如图1-21所示。

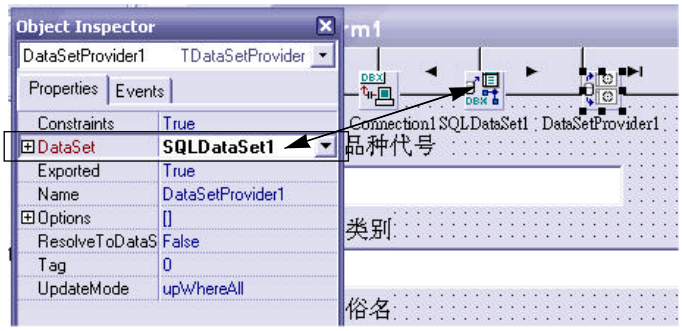


图1-20 设置TDataSetProvider组件的DataSet特性值

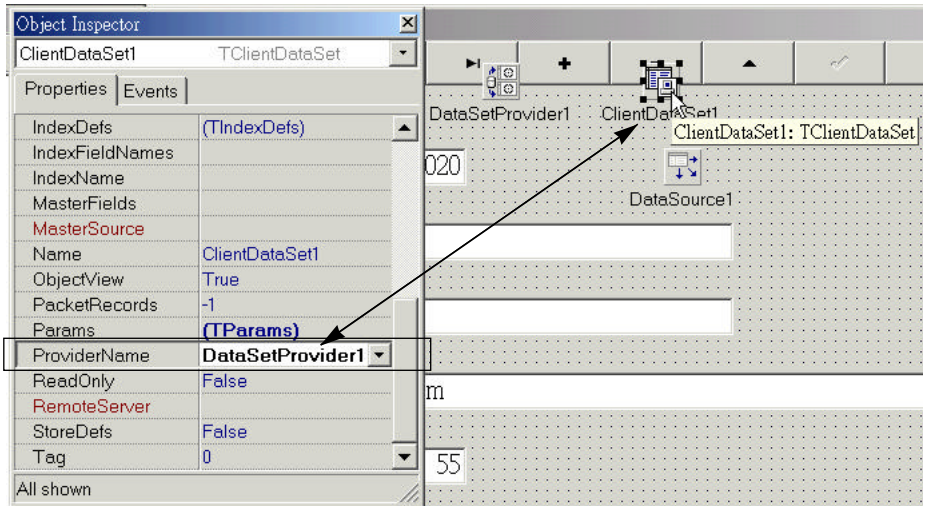


图1-21 设置TClientDataSet组件的ProviderName特性值

现在TClientDataSet组件便可以管理由 TDataSetProvider输出的数据，而TDataSetProvider输出的数据就是它连接的TSQLDataSet从后端数据源中取得的结果数据集。

最后把主窗体中的TDataSource组件的DataSet特性值从原先的TSQLDataSet改为刚才加入的TClientDataSet组件，再把TClientDataSet组件的Active特性值设置为True，如此一来TClientDataSet便从TDataSetProvider取得数据，并且显示在主窗体中的数据感知组件中。

现在请执行这个范例应用程序，读者会看到类似图 1-22的画面，从图1-22中可以看到在使用了 TDataSetProvider和TClientDataSet之后，范例应用程序便可以使用TDBNavigator任意地移动目前的记录，再也不会出现错误了。因此使用 DataSnap技术之后，dbExpress单向游标的限制便已经被克服了。

此外TDBNavigator中的新增、修改和删除按钮似乎也可以工作了，不像图 1-12中

一样被暂停使用。读者可以在范例应用程序中试着修改或是删除数据，然后点击“˘”按钮把数据更新回去。这样做似乎是正确的，但是如果读者结束范例应用程序，然后再次执行范例应用程序，会发现原先修改或是删除的数据又会出现在范例应用程序中，这代表刚才修改和删除的动作并没有真正对数据发生作用，这是为什么呢？

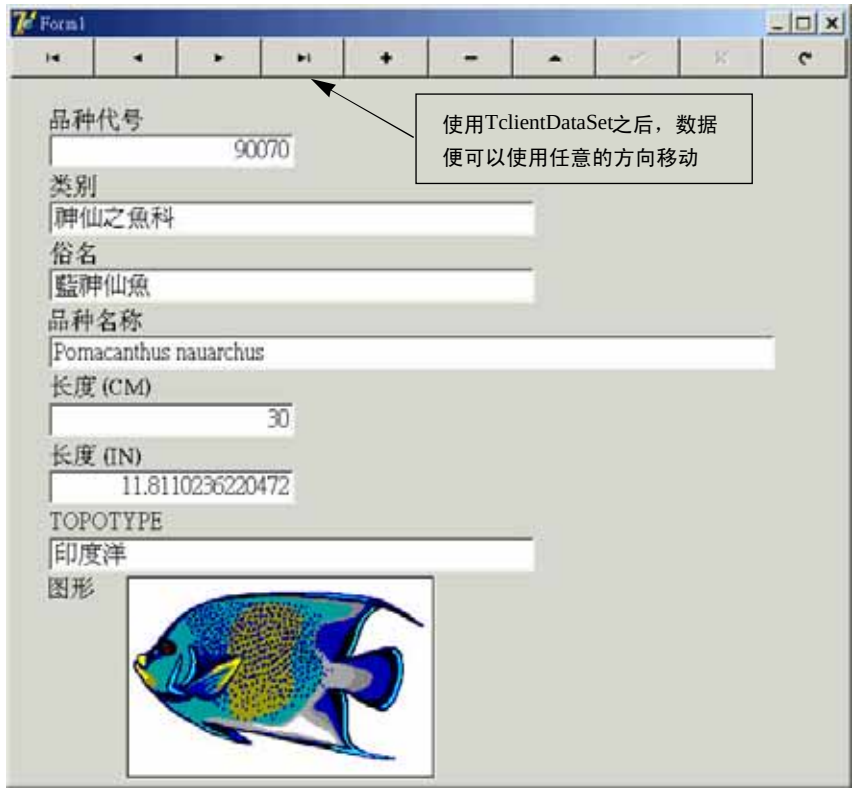


图1-22 执行修改过的范例程序

这是因为在使用 DataSnap的应用程序中，当应用程序修改数据并且调用TClientDataSet的Post方法或是点击TDBNavigator“˘”按钮更新数据时，事实上只是把数据更新回图 1-18中由DataSnap管理的缓存内存中，数据并没有真正更新回后端数据源中。要真正地把修改的数据更新回数据源中，程序员必须再调用TClientDataSet的ApplyUpdates方法。

TClientDataSet的ApplyUpdates方法会把图1-18中由DataSnap管理的缓存内存中所有已经被修改的数据（包括新增、修改和删除的数据）一起更新回后端数据源中。TClientDataSet事实上是通过它连接的 TDataSetProvider组件自动产生 SQL语句帮助程序员更新数据的，在稍后的章节中会详细地说明这个流程。下面是 ApplyUpdates方法的声明原型：

```
function ApplyUpdates(MaxErrors: Integer): Integer; virtual;
```

ApplyUpdates方法接受一个整数类型的参数， MaxErrors。MaxErrors代表当TDataSetProvider自动更新数据时，程序员所允许发生的错误次数。刚才本书已经说明了，当调用ApplyUpdates方法时，DataSnap会一起更新在缓存内存中应用程序从上次调用ApplyUpdates方法之后到这次调用 ApplyUpdates方法之间被修改的所有数据，因此这可能会有许多修改的数据，而 MaxErrors就代表在更新这些数据时程序员允许发生错误的次数。如果 ApplyUpdates在更新数据时发生了超过 MaxErrors指定的数量的错误，那么这整个更新动作便会被回滚。相反的，如果发生的次数小于或是等于MaxErrors，那么成功更新的数据仍然会被更新到数据源中，至于没有成功更新的数据则可以让程序员通过错误事件处理函数来决定如何处理这些失败的数据。本书在稍后讨论异常处理的章节中会详细说明这个问题。通常传递给 ApplyUpdates方法的MaxErrors参数是0，代表不允许发生任何更新错误。或是传递- 1，代表不管发生多少错误都没有关系，先把能够成功更新的数据更新回数据源中。

因此，现在要真正把范例应用程序修改的数据更新回数据源中便非常简单了，只需要在范例应用程序中调用 TClientDataSet的ApplyUpdates方法即可。现在请在主窗体中加入一个TButton，设置它的Caption特性值为 ApplyUpdate，如图1-23所示。

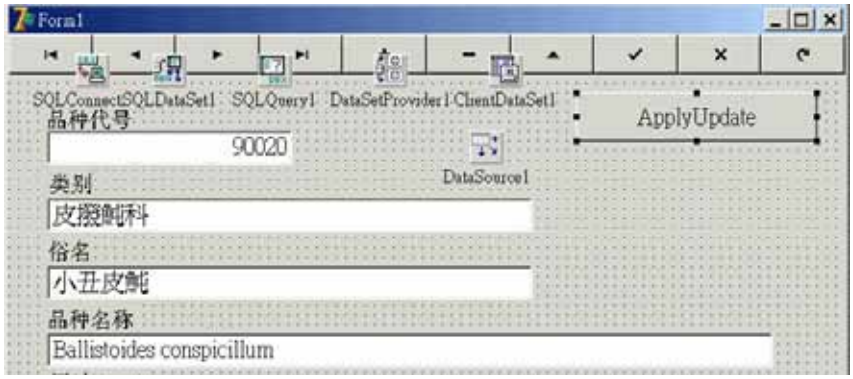


图1-23 在范例主窗体中加入TButton组件

接着在TButton的OnClick事件处理函数中编写如下的程序代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientDataSet1.ApplyUpdates(0);
end;
```

上面的程序代码调用了 TClientDataSet的ApplyUpdates方法，并且传入0代表不允许发生任何错误，如果在修改数据时发生了任何问题，那么 DataSnap便会产生异常错误让程序员得以处理错误状况，在稍后的章节中会说明如何处理 DataSnap的异常状况。

请再次执行范例应用程序，试着修改数据，然后点击主窗体中的 ApplyUpdate按

钮，读者便会发现现在数据真的被更新回后端数据源中了。

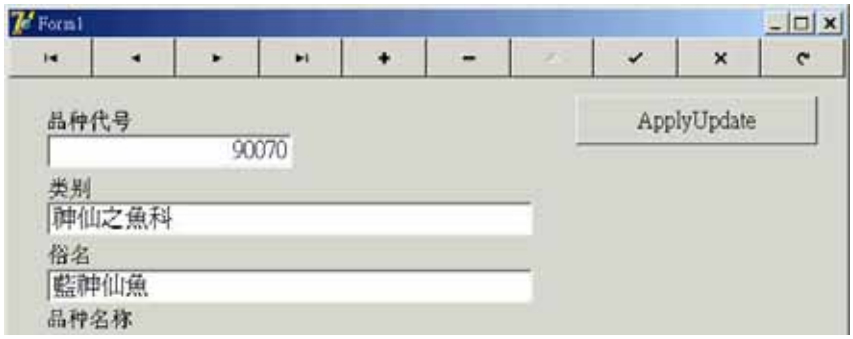


图1-24 范例程序现在可修改数据了

当然，如果读者希望用户无需额外点击按钮便能够把数据更新回数据源中，那么读者可以在TClientDataSet的AfterPost事件处理函数中直接调用TClientDataSet的ApplyUpdates方法，如下所示：

```
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
  ClientDataSet1.ApplyUpdates(0);
end;
```

不过这样写应用程序会降低性能，读者可以让TClientDataSet的修改数据达到一定的数量之后再调用ApplyUpdates方法，一次更新所有的数据。如此做比较有效率，在稍后的章节中会说明如何实现这种功能。

通过这个范例，读者可能认为使用dbExpress和DataSnap的应用程序似乎比较麻烦，然而事实上并非如此，因为在下一小节讨论TSimpleDataSet时读者便会知道我们可以通过TSimpleDataSet简化这些工作。

使用dbExpress和DataSnap比传统的开发方式有许多的优点，例如这样做比传统使用BDE的应用程序有更好的性能，也可以轻易地把应用程序改为分布式的N层应用系统，也可以把应用程序移植到Linux上。这些好处都是传统的开发方式无法轻易达成的，在读者熟悉了dbExpress和DataSnap之后，可能就不再想使用传统的数据库开发方式了。

1.4.2 使用TSimpleDataSet组件

在前一小节中讨论了使用TSQLQuery加上TDataSetProvider和TClientDataSet组件开发应用程序的方式。由于在使用dbExpress开发数据库应用程序时一定需要使用这些组件的组合，因此Delphi 7的DataSnap组件面板便提供了一个新的组件TSimpleDataSet来帮助程序员简化使用dbExpress开发数据库应用程序的步骤。

简单的说，TSimpleDataSet组件就等于TSQLQuery加TDataSetProvider和

TClientDataSet组件，因此程序员只需要使用 TSQLClientDataSet一个组件便可以开发能够修改数据的应用程序。现在就来说明如何使用 TSimpleDataSet组件。

首先点击Delphi的File|New|Application菜单建立一个新的Delphi应用程序，然后按照1.2小节说明的方式连接InterBase数据库。在主窗体中放入位于dbExpress选项卡的TSimpleDataSet组件，请读者注意图1-25显示了TSimpleDataSet已经在内部自动建立了TSQLDBConnection和TDataSet组件，并且将它们命名为 InternalConnection和InternalDataSet，这样程序员就可以直接使用 TSimpleDataSet的InternalConnection和InternalDataSet组件连接和处理数据，而不需要再额外使用 TSQLDBConnection和TSQLQuery了。

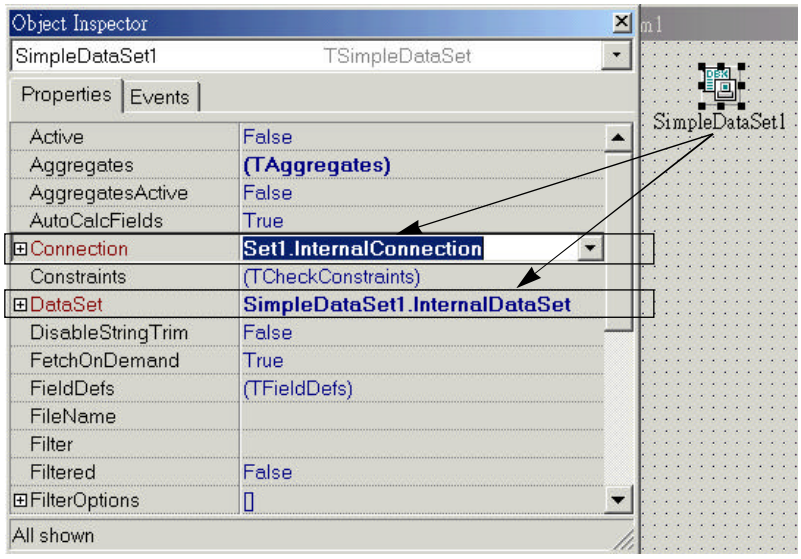


图1-25 TSimpleDataSet组件会自动在内部建立TSQLConnection和DataSet组件

现在我们可以直接在对象检视器中展开 TSimpleDataSet的InternalConnection和InternalDataSet的特性值并且设置到 CHINESEDEMO数据库的连接，再将TSimpleDataSet的DataSet\CommandText特性值设置为从BIOLIFE数据表中取得数据，如图1-26所示。最后再将TSimpleDataSet的Active特性值设置为True以便从数据源中取得数据。

接着在主窗体中放入TDataSource组件，将它的DataSet特性值设置为刚才加入的TSimpleDataSet组件，放入TDBNavigator组件，将它的DataSource特性值设置为刚才加入的TDataSource。最后双击TSimpleDataSet组件激活它的字段编辑器，加入所有的字段对象，再拖曳这些字段对象到主窗体中以建立数据感知组件。

最后，在主窗体中同样加入一个 TButton，将它的 Caption特性值设置为ApplyUpdate，并且在它的 OnClick事件处理函数中调用 TSimpleDataSet组件的

ApplyUpdates方法，把修改的数据更新回数据源中。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  SimpleDataSet1.ApplyUpdates(0);
end;
```

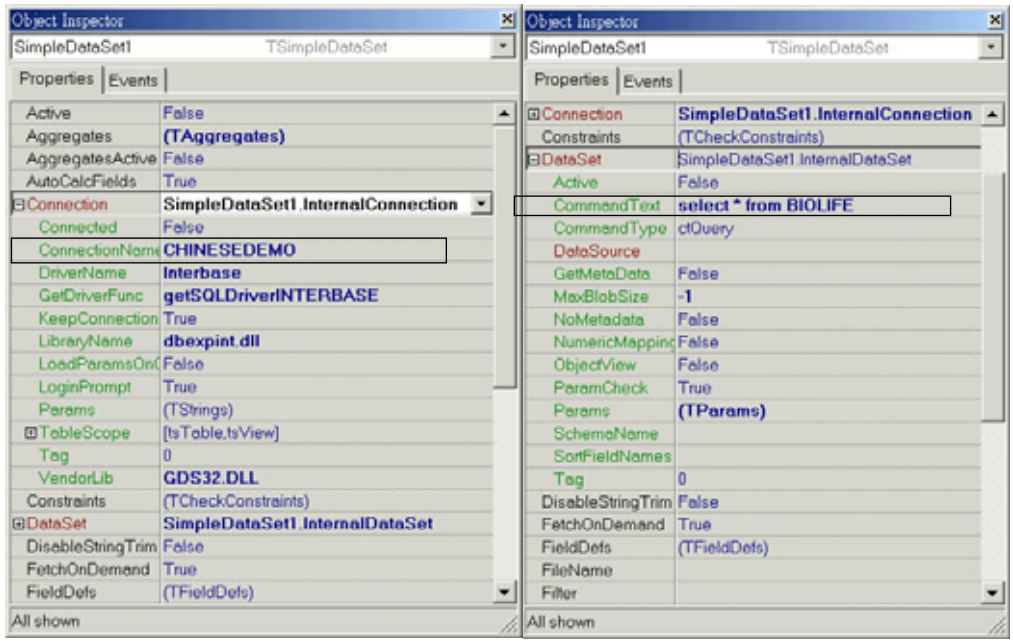


图1-26 使用对象检视器直接设置TSimpleDataSet的InternalConnection和InternalDataSet的特性值

此时范例应用程序应该看起来如图 1-27所示。

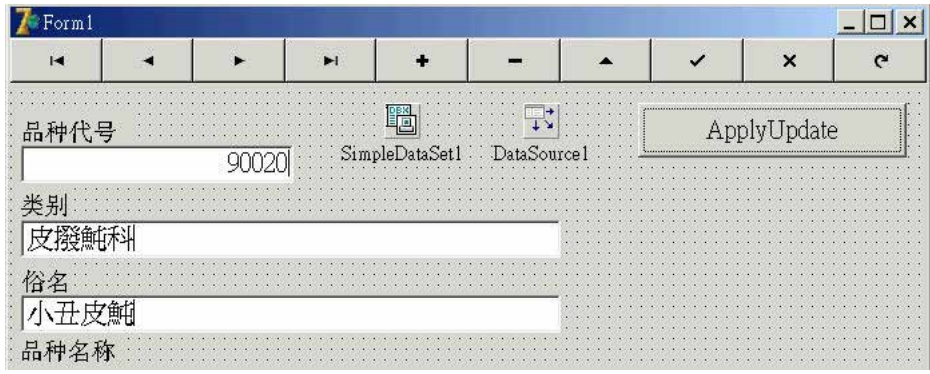


图1-27 执行使用TSimpleDataSet的范例程序

现在读者就可以执行这个使用 TSimpleDataSet的范例应用程序了，图 1-28便是这

个新的范例应用程序执行的画面。这个新的范例应用程序与 1.3.1小节开发的范例应用程序在功能上是一模一样的，只是使用的组件不一样，使用 TSimpleDataSet比使用TSQLDataSet加TDataSetProvider和TClientDataSet组件方便得多，在开发一般应用程序的时候，直接使用TSimpleDataSet是比较方便的。

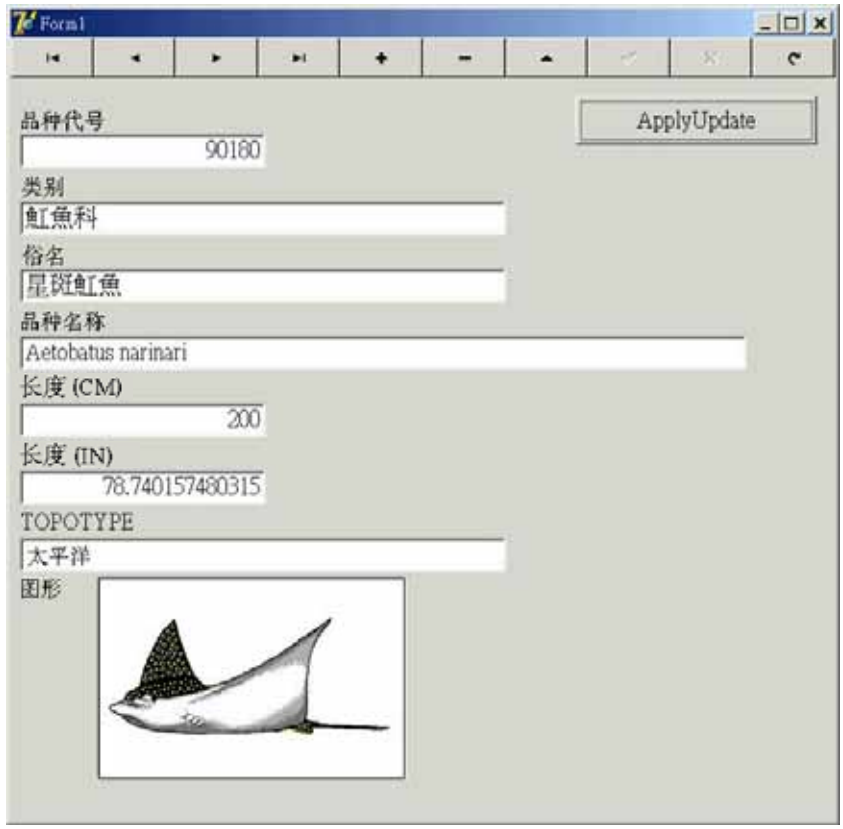


图1-28 使用TSimpleDataSet范例应用程序执行的画面

也许读者会问，既然Delphi 7提供的TSimpleDataSet组件可以更方便地开发数据库应用程序，那么还会需要使用 TSQLQuery加TDataSetProvider和TClientDataSet组件这种比较麻烦的方式吗？答案是肯定的，在后面讨论性能的章节中读者便会了解在高性能要求的应用中，TSQLDataSet加TDataSetProvider和TClientDataSet组件还是非常有用的。

TSimpleDataSet是Delphi 7才加入的新组件，在 Delphi 6和Kylix 2/3中的dbExpress则是使用TSQLClientDataSet。Delphi 7使用TSimpleDataSet代替TSQLClientDataSet的原因除了TSimpleDataSet更容易使用之外，TSimpleDataSet的性能也比TSQLClientDataSet好了许多，比较适合用来开发简易或是2层的数据库应用系统，有关 TSQLClientDataSet效率的问题会在稍后的

章节中说明。但是如果读者已经在 Delphi 6中使用TSQLClientDataSet组件或是需要开发跨平台的dbExpress应用系统，那么仍然可以使用 TSQLClientDataSet。在 Delphi 7中，TSQLClientDataSet是放在Delphi 7安装目录下的\Demos\DB\SQLClientDataSet子目录中，读者可以自行将 TSQLClientDataSet安装到Delphi 7中再使用。

1.5 dbExpress驱动程序的设置

在前面1.2小节中已经说明了 TSQLConnection组件使用的连接信息是来自dbxconnections.ini这个文件的内容。当程序员使用 TSQLConnection建立一个定制的数据库连接时，例如前面建立的 CHINESEDEMO连接，TSQLConnection组件便在dbxconnections.ini文件写入此定制连接的信息。事实上 dbExpress的连接信息是由两个配置文件（configuration file）定义的，它们分别是：

配置文件名称	功能说明
dbxconnections.ini	存储Delphi内建的数据库连接信息以及程序员定义的定制连接信息
dbxdrivers.ini	存储dbExpress支持的实际数据库连接信息。例如数据库 dbExpress驱动程序的进入点函数名称、用户名、密码、使用的 Transaction级别、数据库服务器名称以及数据库名称等，属于每一个数据库特定的信息

dbxdrivers.ini文件中定义的基本上是每一个特定数据库使用的连接信息。而当程序员在TSQLConnection中建立新的数据库连接信息时，TSQLConnection会到dbxdrivers.ini文件中找到此特定数据库的连接信息作为定制连接信息的模板（template），再由程序员修改其中特定的数值，例如用户登录名和密码等。因此我们可以说dbxdrivers.ini提供了每一个dbExpress支持的数据库的模板信息。

当程序员建立了定制的数据库连接之后，TSQLConnection便会在dbxconnections.ini文件中写入此连接信息以便让 dbExpress驱动程序在实际连接时作为连接信息之用。例如，在前面我们建立的 CHINESEDEMO在dbxconnections.ini中存储了如下的信息：

```
[CHINESEDEMO]
DriverName=Interbase
BlobSize=-1
CommitRetain=True
Database=e:\LeeWei\Books\Delphi7\Datas\CHINESEDEMO.GDB
Password=masterkey
RoleName=RoleName
ServerCharSet=ASCII
SQLDialect=1
Interbase TransIsolation=ReadCommited
```

```
User_Name=sysdba
WaitOnLocks=True
```

上面的连接信息中清楚地标明了 CHINESEDEMO是连接到InterBase数据库的定制连接，因此当dbExpress应用程序激活时看到此参数选项便会到 dbxdrivers.ini中搜寻InterBase的dbExpress驱动程序细节。而在dbxdrivers.ini中则有如下的信息：

```
[Interbase]
GetDriverFunc=getSQLDriverINTERBASE
LibraryName=dbexpint.dll
VendorLib=GDS32.DLL
BlobSize=-1
CommitRetain=True
Database=database.gdb
Password=masterkey
RoleName=RoleName
ServerCharSet=ASCII
SQLDialect=1
Interbase TransIsolation=ReadCommited
User_Name=sysdba
WaitOnLocks=True
```

从上面的信息中可以看到对于 InterBase的连接，dbExpress需要加载dbexpint.dll这个DLL文件，而且 dbexpint.dll函数库的进入点是 getSQLDriverINTERBASE。dbExpress在使用LoadLibrary加载dbexpint.dll之后，就可以使用 GetProcAddress取得getSQLDriverINTERBASE函数的地址，调用 getSQLDriverINTERBASE就可以取得dbExpress的ISQLConnection接口，接着就可以连接到 InterBase数据库了。在后面的第13章中会说明 dbExpress的实现原理，读者可以参考其中说明的内容。而dbexpint.dll则调用了InterBase的客户端API GDS32.DLL函数库来真正与InterBase连接交互。

Delphi 7已经正式支持MS SQL Server 2000以及Informix，但是在前面图1-4中我们却只看到了DB2、InterBase、MYSQL以及Oracle数据库。那么我们如何建立连接到MS SQL Server 2000或是Informix的定制数据库连接呢？如果读者了解了刚才说明的dbxconnections.ini和dbxdrivers.ini这两个配置文件的功能，就可以知道这是非常容易的。

首先，读者可以开启 dbxdrivers.ini这个dbExpress驱动程序模板文件，便会在文件一开始的地方看到如下的信息：

```
[Installed Drivers]
DB2=1
Interbase=1
MYSQL=1
Oracle=1
```

这四个数据库的标志都设置为 1，因此这四个数据库连接信息就是我们在图 1-29 中看到的dbExpress支持的四个数据库。因此要让 TSQLConnection也在Delphi的集成开发环境中允许程序员使用 MS SQL Server和Informix，我们只需要在Oracle=1下面加入如下二行程序代码即可：

```
Informix=1
MSSQL=1
```

接着存储dbxdrivers.ini，再重新使用 TSQLConnection建立新的数据库连接，那么读者就可以看到如图 1-30所示的画面，现在我们也可以建立 MS SQL Server和Informix的定制连接了。

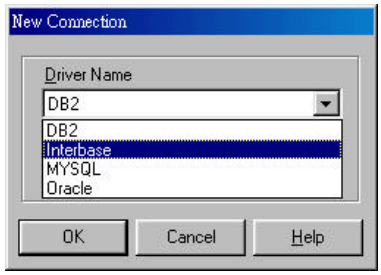


图1-29 在默认情况下TSQLConnection只提供了四个数据库的连接信息

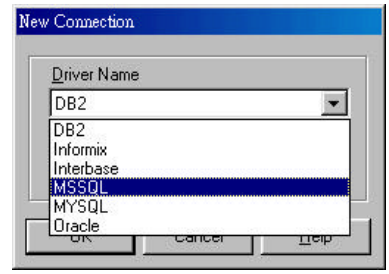


图1-30 在dbxconnections.ini中加入了MS SQL Server和Informix的驱动程序标志之后便可在TSQL-Connection中看到新的内置数据库连接

如果点击图1-30中的MSSQL，那么就可以在dbExpress Connections对话框中看到TSQLConnection从dbxdrivers.ini配置文件搜寻用于MS SQL Server的连接模板信息（见图1-31）。之后我们就可以修改其中的特性值以连接到正确的 MS SQL Server了。

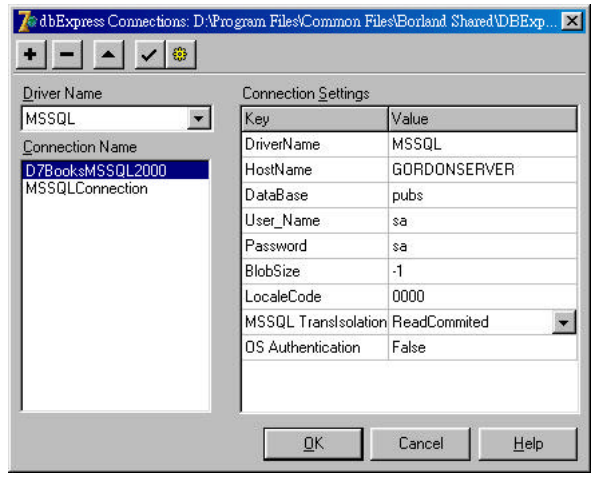


图1-31 设置使用MS SQL Server的定制数据库连接之后，TSQLConnection直接从dbxdrivers.ini中取得MS SQL Server特定的连接信息

最后，我们开启TSQLConnection的View Driver Settings对话框，就可以清楚地看到每一个dbExpress驱动程序使用的特定数据库厂商的函数库。例如，从图 1-32中我们可以确定Delphi 7的MS SQL Server驱动程序是直接使用OLEDB来连接，而不是使用ADO。

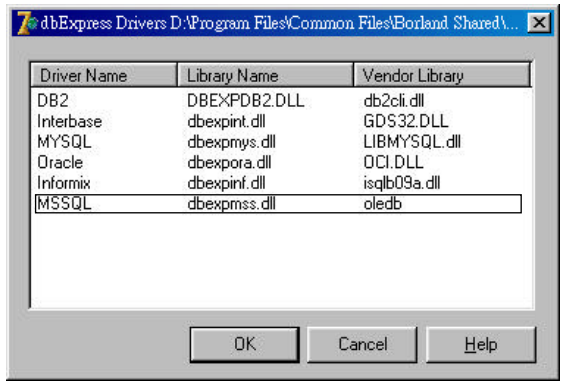


图1-32 在TSQLConnection的View Driver Settings对话框中现在可以看到各个dbExpress驱动程序使用的底层厂商驱动程序，例如 MS SQL Server是使用OLEDB连接MS SQL Server数据库的

其实Delphi也支持连接PostGreSQL的驱动程序，读者可以使用本小节讨论的过程为Delphi 7加入连接PostGreSQL的dbExpress信息。

1.6 结论

本章以范例的形式说明了如何使用 dbExpress快速开发数据库应用程序，让读者能够马上使用dbExpress组件来编写应用程序。对于熟悉原先 BDE组件的程序员来说，新的dbExpress技术似乎比较麻烦，需要使用数个组件才能开发出能够修改数据的数据库应用程序。但是通过使用新的 TSimpleDataSet组件，dbExpress开发模式就和以往的BDE一样方便。对于新接触 Delphi的程序员来说，dbExpress组件也是非常方便和直观的。

dbExpress技术包含dbExpress组件、dbExpress驱动程序以及DataSnap技术，本章讨论的内容只是dbExpress的冰山之一角，读者除了使用 dbExpress组件开发应用程序之外，也需要熟悉许多技术以及如何巧妙地使用 TSimpleDataSet的技术来处理数据。这些技术都会在后面的章节中一一讨论。

本章除了说明如何使用 dbExpress开发应用程序之外，也说明了 dbExpress的原理和结构，让读者能够掌握 dbExpress的核心概念，这有助于了解随后章节的内容。当然，在稍后的章节中会继续说明更多重要的 dbExpress概念和技术。

第2章　使用dbExpress组件

在上一章中讨论了dbExpress的概念以及如何使用dbExpress快速建立应用程序。从本章开始将逐渐让读者了解更多有关使用dbExpress的细节以及如何更精确地使用dbExpress中的组件来开发应用程序。

由于完整的dbExpress技术需要集成Delphi的MIDAS技术，也就是从Delphi 6/7开始称为DataSnap的技术。因此在本章以及随后的章节中也将会一并讨论许多DataSnap技术。DataSnap不但可以搭配dbExpress开发单机以及主从结构的应用系统，也可以结合MTS/COM+等组件模型开发分布式多层应用系统，因此DataSnap在Delphi以及Linux上的Kylix中是非常重要的技术，读者一定要切实掌握这项技术。

现在就让我们从dbExpress中最重要的数据处理组件TSimpleDataSet开始学习dbExpress和DataSnap技术。

2.1　使用TSimpleDataSet组件

在第1章中，本书已经说明了在使用Delphi 7的dbExpress开发数据库应用程序时最基本的方式是使用TSQLConnection和TSimpleDataSet组件，如此一来才能够处理和修改数据，因此读者必须了解如何使用TSimpleDataSet组件才能正确地使用dbExpress技术。

TSimpleDataSet组件有许多重要的特性和事件让程序员能够精确地控制处理数据的行为。TSimpleDataSet的许多特性值都会影响它如何访问和处理数据，有一些特性值也和性能有重要的关系。因此这些特性是读者一定要了解和能够正确使用的，下面的小节便讨论了这些重要的特性以及相关的事件。

1. DataSet\CommandType

当程序员使用TSimpleDataSet组件时，必须要决定如何使用它来访问数据。TSimpleDataSet组件使程序员可以执行各种SQL语句来访问和处理数据，也使程序员可以使用它来执行后端数据源中的存储过程（Stored Procedure），或是直接使用它来取得数据源中某一个数据表中的数据，因此TSimpleDataSet组件的功能可以说是非常丰富的。

这些决定如何使用TSimpleDataSet的特性便是DataSet\CommandType，根据程序员为DataSet\CommandType设置的特性值可以决定TSimpleDataSet的执行行为。下面的表格列出了DataSet\CommandType可以设置的特性值以及这些特性值的意义：

CommandType特性	意 义
ctQuery	执行SQL语句
ctStoredProc	执行后端数据源中的存储过程
ctTable	访问指定的数据表中的所有数据

TSimpleDataSet的DataSet\CommandType的默认值是ctQuery，主要是执行SQL语句，如果程序员需要使用TSimpleDataSet执行存储过程，那么必须设置DataSet\CommandType为ctStoredProc。

2. Active特性

Active特性值可以让TSimpleDataSet组件执行程序员在DataSet\CommandText特性值中指定的SQL语句（当DataSet\CommandType是ctQuery时），或是取得DataSet\CommandText特性值指定的数据表中的所有数据（当DataSet\CommandType是ctTable时）。不过设置Active特性值只能让TSimpleDataSet执行会返回结果数据集的SQL语句，对于不返回结果数据集的SQL语句，程序员必须调用TSimpleDataSet的Execute方法。下面的表格总结了TSimpleDataSet组件设置Active特性值以及调用Execute方法的意义：

执行SQL语句	意 义
Active特性值	执行会返回结果数据集的SQL语句，例如Select...的SQL语句
Execute	执行不返回结果数据集的SQL语句，例如Delete、Update、Insert或是DDL语句

设置Active特性值为True就等于调用TSimpleDataSet的Open方法，设置为False就等于调用TSimpleDataSet的Close方法。

3. PacketRecords特性

TSimpleDataSet组件的PacketRecords是一个非常重要的特性值，因为它控制了TSimpleDataSet组件如何访问数据。TSimpleDataSet组件的PacketRecords基本上可以设置成下列表格中的数值。不同的PacketRecords特性值代表了不同的数据访问行为。

PacketRecords特性	意 义
-1	一次从后端数据源中取得所有数据
0	取得描述数据源的元数据信息
正数	一次只取得指定数量的记录

PacketRecords的默认值是-1，代表当TSimpleDataSet组件开启后它会一次把后端数据源中的所有数据读到客户端。对于数据量少的数据表来说这种访问行为没有什么关系。但是请想想，对于数据量大的数据表，例如拥有数万个记录的数据表，这样做不但没有效率，而且会造成沉重的网络负荷。这是程序员应该尽量避免的。

因此对于拥有大量数据的数据源来说，程序员应该设置PacketRecords特性值为一个正数，例如20。这就代表当TSimpleDataSet开启时它只会访问20个记录，就暂

时停止。如果用户浏览或处理到了20个之后的记录，那么TSimpleDataSet便会自动从后端数据源中取得下20个记录。这种访问行为非常适合一般的使用，因为用户一次也不可能浏览或处理太多的数据，因此让TSimpleDataSet以分段的方式访问客户端需要的数据不但可以加快应用程序的反应时间，减轻客户端的负荷，也可以降低网络的数据传递量。不过程序员也不能将PacketRecords特性值设置为太小的数值，因为这会造成TSimpleDataSet频繁地从后端数据源访问数据，反而会降低性能，增加网络的往返次数。一般来说，将PacketRecords设置为100左右的数值是比较好的，当然读者在实际应用中需要根据同时使用客户端的人数自行调整PacketRecords数值，在稍后讨论性能的章节中会再说明。

最后，如果程序员将PacketRecords设置为0，那么就代表要从后端数据源中取得描述结果数据集的元数据。一般来说，当TSimpleDataSet/TClientDataSet第一次开启后端的结果数据集时，都会先从数据源中取得元数据，建立客户端结果数据集的结构以便存储从后端结果数据集取得的数据。

> 所谓元数据（metadata）是指描述数据的信息。例如描述数据表的元数据包括了这个数据表所有的字段名称、字段类型、字段长度等信息。通常客户端能够根据这些元数据正确地建立客户端数据集，以及自动产生正确的SQL语句。元数据对于程序是否能够有效率和正确地执行有很大的影响。

> TSimpleDataSet/TClientDataSet会不会在第1次开启后端数据源时取得元数据是根据不同的dbExpress驱动程序决定的。不过以目前dbExpress驱动程序优化做得越来越好的状况来看，dbExpress驱动程序都避免进行这个动作以增加性能，在稍后的章节中会详细的讨论。

由于PacketRecords特性值是如此重要，因此程序员在使用组件时必须对于这个特性值进行适当的设置。

4. Data特性

当TSimpleDataSet从后端数据源取得数据之后，这些数据便存储在TSimpleDataSet组件的Data特性值中，因此程序员可以通过访问Data特性值来取得原始数据。当客户端应用程序修改数据时，存储在Data中的数据会被改变，同时所有被客户端应用程序修改的数据也都会暂时存储在Delta特性值中。

5. Delta特性

TSimpleDataSet的Delta特性值存储的是客户端应用程序对于Data特性值中数据的修改。在TSimpleDataSet从后端数据源取得数据时，Delta特性值的内容最初是空白的。但是一旦应用程序修改了数据，那么修改的数据便会暂时存储在Delta中。而当

应用程序调用了TSimpleDataSet的ApplyUpdates方法之后，Delta特性值中的数据便会真正更新回后端数据库中。在ApplyUpdates方法成功执行完毕之后，Delta中的数值便会被清除。程序员可以通过访问Delta中的数值来取得目前被修改的数据。

虽然在一般的应用中程序员也许并不需要直接使用Data和Delta特性值，但是在许多高级的应用中这两个特性值却可以发挥非常大的作用，特别是在处理复杂的数据运算时，在稍后的章节中会有范例介绍如何使用Data和Delta特性。

> 考虑到更大的控制能力以及稍后章节会讨论的性能因素，因此笔者建议各位读者尽量不要直接使用TSimpleDataSet组件，而使用TSQLQuery、TClientDataSet和TDataSetProvider组件。当然，如果只是一般的应用，那么使用TSimpleDataSet是非常便利的。因此本小节随后的范例都将同时使用TSQLClientDataSet组件和这三个组件来说明。

TSimpleDataSet组件还有许多重要的特性和事件，在稍后的章节中将会一一介绍这些方法、特性和事件。现在就让我们开始通过范例来学习如何使用TSimpleDataSet组件。

2.1.1 使用动态SQL语句处理数据

在第1章中已经说明了如何使用TSimpleDataSet组件，我们通过将SQL语句设置到TSimpleDataSet的DataSet\CommandText特性值中便可以从后端数据源中取得数据。本小节将进一步说明如何使用动态SQL语句来访问和处理数据。

步骤1：建立数据模块和dbExpress组件

首先在Delphi集成开发环境中点击File|New|CLX Application，再点击File|New|Other...菜单，在New Items对话框中点击CLX Data Module图标，建立一个CLX数据模块，如图2-1所示。

在空白的CLX数据模块中加入一个TSQLConnection组件，将它的Name特性值设置为scnnDemo。双击TSQLConnection组件以激活它的组件编辑器，并且连接到InterBase范例数据库D7Books.GDB，如图2-2所示（D7Books使用Add Connection建立的InterBase连接，请读者自行建立，并且设置ServerCharSet特性值为空白）。

接着在CLX数据模块中放入一个TSimpleDataSet组件，设置它的Name特性值为scdsBooks，设置它的DBConnection特性值为scnnDemo。虽然TSimpleDataSet会自动在内部建立TSQLConnection，但是我们仍然可以强迫TSimpleDataSet使用外部的TSQLConnection。

再设置scdsBooks的DataSet\CommandText特性值为select * from BOOKS，设置Active特性值为True，从BOOKS数据表中取得所有的数据。最后放入另外一个

TSimpleDataSet组件，设置它的Name特性值为scdsPublishers，设置DBConnection特性值为scnnDemo，再设置DataSet\CommandText特性值为select * from PUBLISHERS where VID = :VID。从这个范例可以看到使用外部TSQLConnection的好处，因为在默认情况下scdsBooks和scdsPublishers都会在内部自行建立TSQLConnection，但是既然scdsBooks和scdsPublishers都连接到相同的数据库，那么我们就不需要建立两个TSQLConnection浪费资源。

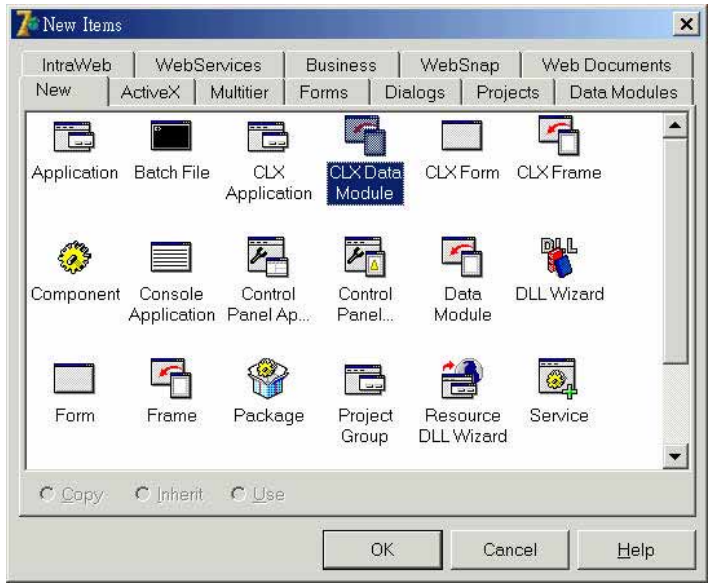


图2-1 建立CLX数据模块

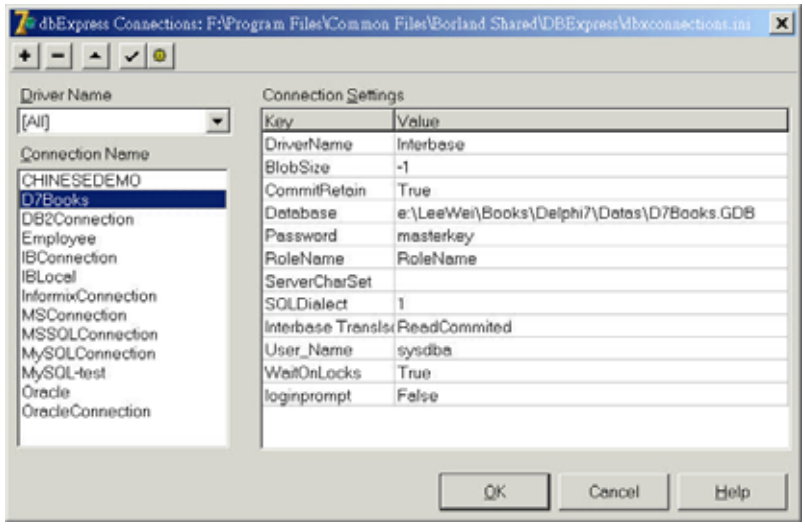


图2-2 激活TSQLConnection组件的组件编辑器

scdsPublishers在DataSet\CommandText特性值中使用的SQL语句称为动态SQL，因为在这个SQL语句中定义了一个参数:VID。这个参数的数值会在应用程序执行时动态地传入，再根据这个参数值从PUBLISHERS数据表中取得VID字段拥有传入的参数值的数据。当然，在这个SQL语句中参数名称VID可以命名为其他名称，例如:ID1或是:PID。在SQL语句中动态参数是以：

:参数名称

形式定义的，而且在SQL语句中可以使用多个动态参数。

dbExpress组件的TSQLDataSet、TSQLQuery、TSQLStoredProc和TSimpleDataSet都允许程序员使用有动态参数的SQL语句来访问数据。此时范例应用程序的数据模块看起来如图2-3所示。

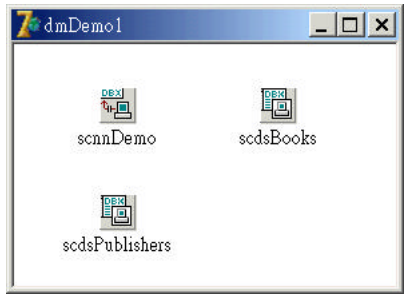


图2-3 范例应用程序的数据模块

由于scdsPublishers组件定义了动态参数，因此它会自动分析SQL语句，并且从其中取得所有动态参数，然后存储在TSimpleDataSet组件的DataSet\Params特性值中。我们可以点击scdsPublishers组件，然后在对象检视器中双击它的Params特性，那么Delphi便会显示scdsPublishers定义的动态参数。例如，图2-4便是双击scdsPublishers组件的Params特性之后，Delphi显示的参数对话框，其中列出了VID这个动态参数。此时程序员可以点击这个动态参数，然后在对象检视器中设置此动态参数的特性值。图2-4显示了VID是一个输入参数，我们需要设置它的数据类型是字符串（ftString）。

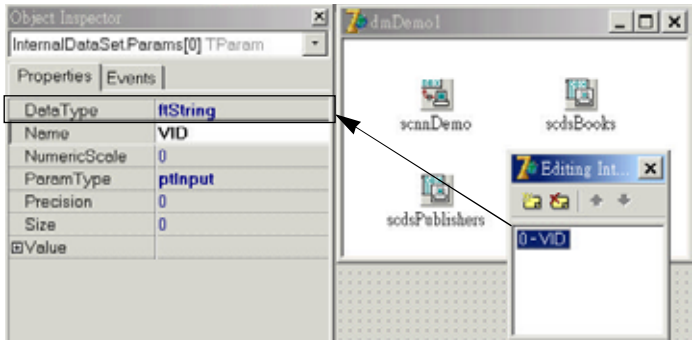


图2-4 scdsPublishers组件在Params特性值中定义的动态参数

步骤 2：建立范例应用程序的主窗体

回到范例应用程序的主窗体，在其中放入 TDataSource，并且设置它的DataSet特性值为步骤1建立的scdsBooks组件（注:读者必须在主窗体中先use步骤1的数据模块，即点击File|Use Unit...选项，选择步骤 1的dmDemo1数据模块），再放入TDBNavigator和TDBGrid组件，设置它们的 DataSource特性值为刚才加入的 TDataSource。最后在主窗体下方放入一个 TPanel组件，此时范例主窗体看起来类似于图 2-5。

图2-5 范例应用程序的主窗体

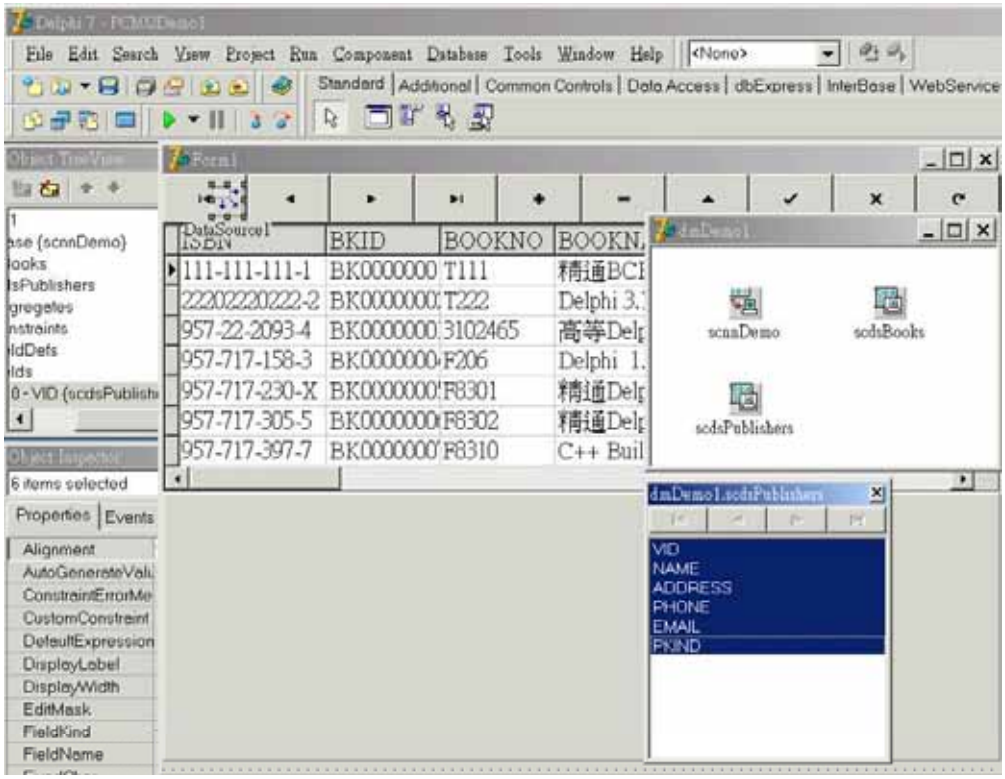
现在我们希望当用户浏览一本书籍时，能够同时显示出版这本书的出版商信息，因此我们需要显示 PUBLISHERS数据表中的数据。请开启数据模块，双击scdsPublishers组件以激活它的字段编辑器，接着把 PUBLISHERS数据表的所有字段对象拖曳到刚才加入的TPanel组件中。此时主窗体如图 2-6所示。

接下来的工作就是实现此范例应用程序，它需要的功能就是当用户移动书籍数据时，把出版此书的出版商信息显示在下方的数据感知组件中。

> 如果读者建立 CLX类型的应用程序，那么可能无法以拖曳的方式建立图 2-6中的数据感知组件，读者需要自行加入 TDBEdit组件来显示数据。

步骤 3：实现范例应用程序

要实现浏览书籍时同时显示出版商信息的功能，那么我们首先需要根据目前正被浏览的书籍的出版商编号到 PUBLISHERS数据表中搜寻。而在前面建立数据模块时



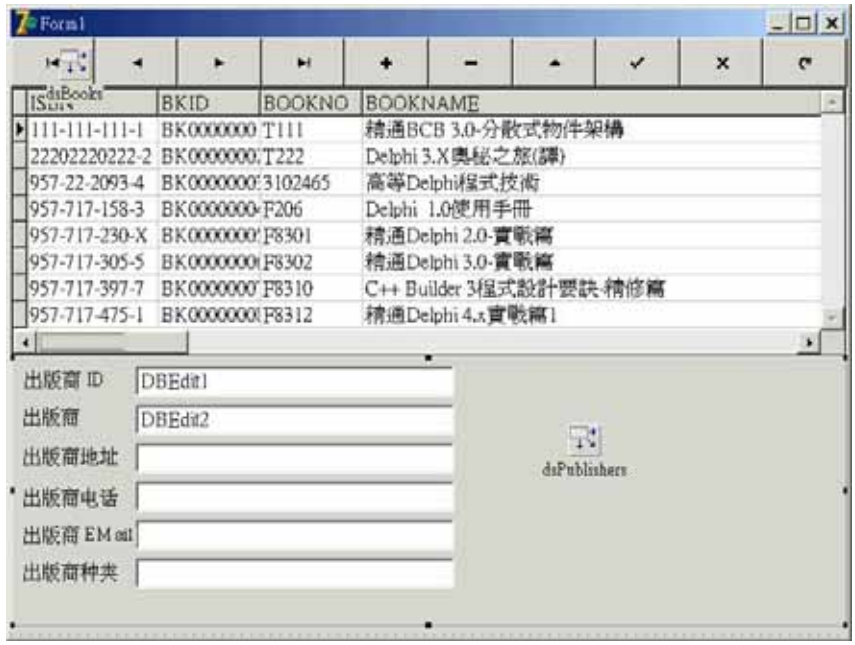


图2-6 在主窗体中加入显示scdsPublishers字段的数据感知组件

scdsPublishers组件的SQL语句正是根据出版商编号从 PUBLISHERS数据表中取得数据的，因此我们只需要把目前正被浏览的书籍的出版商编号传递给 scdsPublishers组件即可。进行这个工作的好时机便是当用户将目前的记录位置移动到其他记录时，因此我们可以在scdsBooks的AfterScroll事件处理函数中进行这个工作。

请在数据模块中点击 scdsBooks，双击它的AfterScroll事件，并且编写如下的程序代码：

```
procedure TdmDynamicSQL.scdsBooksAfterScroll(DataSet: TDataSet);
begin
  try
    Self.scdsPublishers.Active := False;
    Self.scdsPublishers.DataSet.Params.ParamByName('VID').Value :=
      Self.scdsBooks.FieldByName('VID').Value;
    Self.scdsPublishers.Active := True;
  except
    on Exception do;
  end;
end;
```

上面的程序代码首先把目前书籍的出版商 ID传入到scdsPublishers定义的动态参数中。为此，我们只需要访问 scdsPublishers的DataSet.Params特性，再调用 DataSet.Params返回的TParam对象的ParamByName方法即可。最后开启scdsPublishers即可取得正确的出版商数据。

步骤 4：执行范例应用程序

现在就可以编译并且执行范例应用程序了，图 2-7就是范例应用程序此时执行的画面。当用户使用窗体上方的 TDBNavigator移动数据时，下方显示出版商的数据感知组件便会显示出正确的出版商数据。

现在我们已经使用 dbExpress组件和动态 SQL语句的方式顺利地取得范例应用程序需要的数据了。接着让我们观察 dbExpress如何处理修改的数据。

步骤 5：取得目前被修改的数据记录数信息

回到主窗体，在窗体中加入一个 TBitBtn组件，设置它的 Caption特性值为“ApplyUpdates”。再加入一个 TStatusBar组件，双击它并且加入一个新的 TStatusPanel组件。接着我们希望主窗体也能够显示书籍的作者名称，因此另外在主窗体中加入一个TLabel组件，设置它的Caption特性值为“作者”，再加入一个TDBEdit组件，此时范例主窗体应该类似于图 2-8。

要在主窗体中显示书籍的作者，我们也需要在书籍被浏览时到 PERFORMERS数据表中搜寻数据，因此请开启数据模块，在数据模块中再加入一个 TSimpleDataSet组件，设置它的 DBConnection特性值为 scnnDemo，设置它的 Name特性值为

scdsAuthor。此时数据模块类似于图2-9。

图2-7 执行范例应用程序的画面

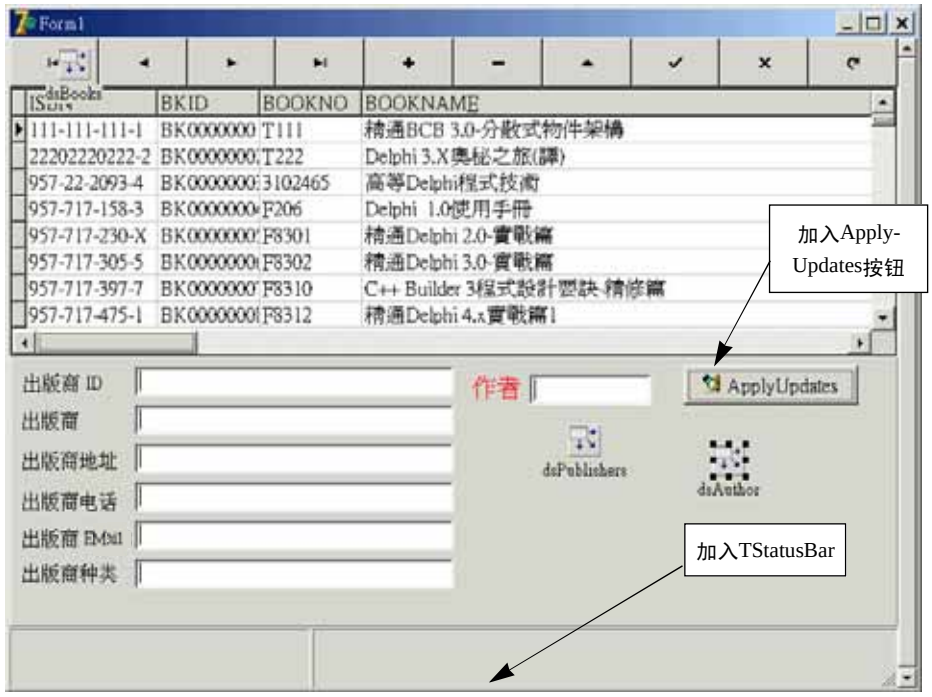


图2-8 在范例应用程序的主窗体中加入TBitBtn组件和TStatusBar组件

接着在前面已经讨论过的 AfterScroll事件处理函数中加入一行调用GetAuthor的程序代码：

```
procedure TdmDemo1.scdsBooksAfterScroll
(DataSet: TDataSet);
begin
  try
    Self.scdsPublishers.Active := False;
    Self.scdsPublishers.Params.ParamByName
    ('VID').Value :=
      Self.scdsBooks.FieldByName('VID').Value;
    Self.scdsPublishers.Active := True;

    GetAuthor;
  except
    on Exception do;
  end;
end;
```

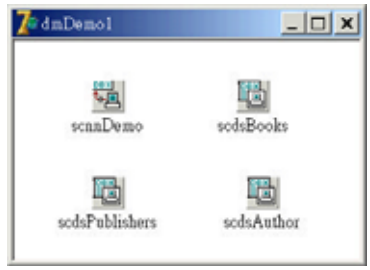


图2-9 在数据模块中加入TSimpleDataSet组件，设置Name特性值为scdsAuthor

而GetAuthor方法则使用了动态组成SQL语句的方式从PERFORMERS数据表中取得数据。下面的程序代码很简单，它从 scdsBooks中取得书籍作者的ID，再通过这个ID组成SQL语句从PERFORMERS数据表中取得正确的数据。

```
procedure TdmDemo1.GetAuthor;
begin
  Self.scdsAuthor.Active := False;
  Self.scdsAuthor.DataSet.CommandText :=
    'select * from PERFORMERS where AID = ' + '''' +
    Self.scdsBooks.FieldByName('AID').Value + '''';
  Self.scdsAuthor.Active := True;
end;
```

现在再执行范例应用程序，我们便可以在主窗体中看到书籍作者的数据了。

> 在这里，本书只是展示程序员也可以使用动态组成 SQL的方式来访问数据，scdsAuthors就是一个范例。但是这里的程序代码并不是很好的，因为它的性能并不好。如果读者使用与前面 scdsPublishers一样的动态参数方法会比较有效率，因为在scdsAuthors的SQL语句中只有作者ID会改变。

在前面本书说过，当用户在应用程序中修改了数据后，这些修改的数据会暂时存储在TSimpleDataSet的Delta特性值中。程序员可以通过访问 TSimpleDataSet的组件ChangeCount特性值来得知目前已经被修改的数据记录数，并且可以根据这个数值来决定是否需要调用 ApplyUpdates方法把修改的数据真正更新回后端数据源中。现在

就让我们在范例应用程序中显示目前被修改的数据记录数。

当TSimpleDataSet调用Post方法把修改的数据暂时存储在Delta特性值中后，它的ChangeCount特性值便会随着更新。因此要得知目前的修改数据记录数，我们可以在scdsBooks的AfterPost事件处理函数中显示 ChangeCount。请点击数据模块中的scdsBooks，并且在它的AfterPost事件中编写如下的程序代码：

```
procedure TdmDemo1.scdsBooksAfterPost(DataSet: TDataSet);
begin
  Form1.ShowChangeCount;
end;
```

上面的程序代码调用了主窗体中的ShowChangeCount方法来显示修改的数据记录数。

接着回到主窗体中双击 ApplyUpdates按钮并且在OnClick事件中编写如下的程序代码。当scdsBooks调用ApplyUpdates方法之后也调用 ShowChangeCount方法再次显示当数据真正被更新回数据源之后客户端暂时内存中 Delta特性值的改变状态。

```
procedure TForm1.bbtnApplyClick(Sender: TObject);
begin
  dmDemo1.scdsBooks.ApplyUpdates(0);
  ShowChangeCount;
end;
```

最后是ShowChangeCount方法的实现，它只是访问scdsBooks的ChangeCount特性值并且在主窗体的TStatusBar组件中显示：

```
procedure TForm1.ShowChangeCount;
begin
  StatusBar1.Panels[0].Text :=
    '目前被修改的数据笔数 : ' + IntToStr
  (dmDynamicSQL.scdsBooks.ChangeCount);
end;
```

现在编译并且执行范例应用程序，图 2-10是执行范例应用程序并且更新两个记录之后的画面。从TStatusBar组件中可以看到ChangeCount特性值正确地记录了目前有两个记录在客户端被修改了。

接着我们点击窗体中的ApplyUpdates按钮调用ApplyUpdates方法把数据真正更新回数据源中。图 2-11显示了当ApplyUpdates方法成功执行完毕之后，客户端记录的暂时修改数据记录数也清除为 0了。

到目前为止，本范例展示了如何使用 dbExpress组件访问数据以及使用动态 SQL在应用程序执行时动态选择应用程序需要的数据。读者现在应该能够使用 dbExpress组件来进行基本的数据处理了。接下来让我们继续通过这个范例应用程序讨论重要的Data和Delta特性。

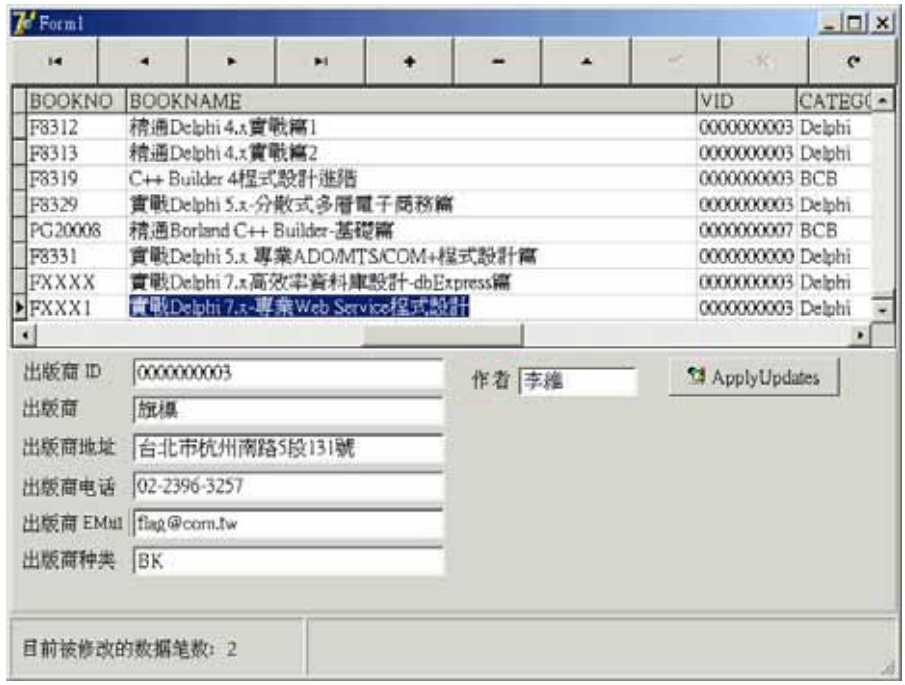


图2-10　执行范例应用程序并且修改数据

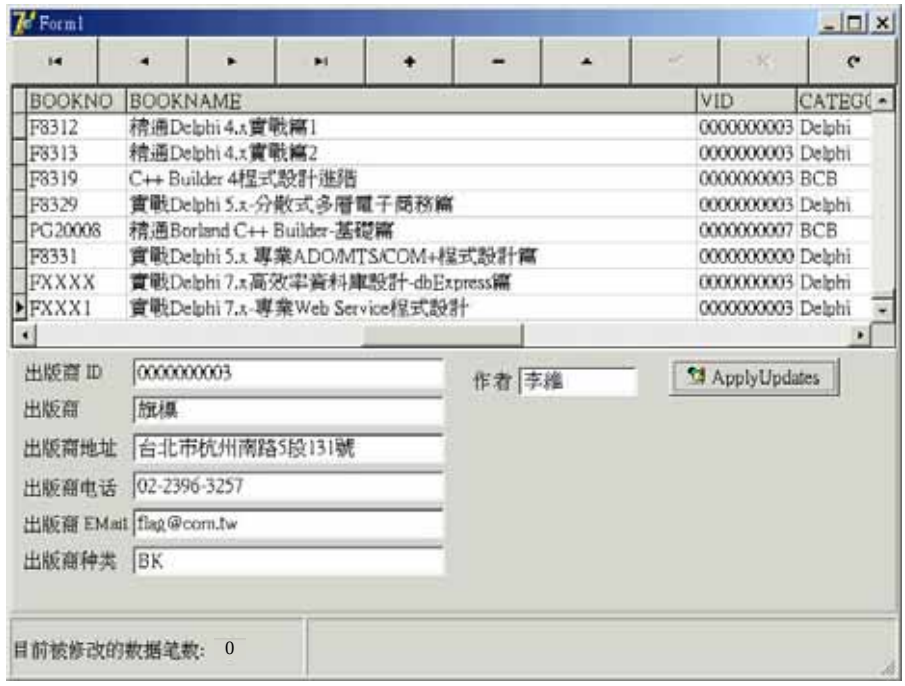


图2-11　范例应用程序调用ApplyUpdates方法后的画面

2.1.2 Data和Delta特性

Data和Delta分别存储了使用dbExpress从数据源中取得的数据以及被用户修改的数据。通过了解Data和Delta特性，程序员能够使用一些高级的技巧来开发应用程序。先让我们观察这两个特性值在范例应用程序执行时的改变情形。

由于Data和Delta特性是OleVariant类型的特性，而且包含在这个 OleVariant数值中的数据结构是 Delphi的DataSnap技术，因此要观察这两个特性的改变，必须使用Delphi的TSimpleDataSet/TClientDataSet组件。我们可以在用户修改了 scdsBooks的数据时把 scdsBooks的Data和Delta特性值指定给另外一个 TSimpleDataSet/TClientDataSet的Data特性，再让这个 TSimpleDataSet/TClientDataSet组件通过TDataSource连接到TDBGrid等数据感知组件，这样就可以观察到 scdsBooks的Data和Delta特性值的变化情形。

为此，请回到范例应用程序并且开启主窗体。在主窗体中加入一个 TClientData-Set组件，再加入一个 TDataSource组件，设置它的 DataSet特性值为刚才加入的TClientDataSet组件。再加入TDBNavigator和TDBGrid组件，设置它们的DataSource特性值为刚才加入的TDataSource组件。此时主窗体如图2-12所示。

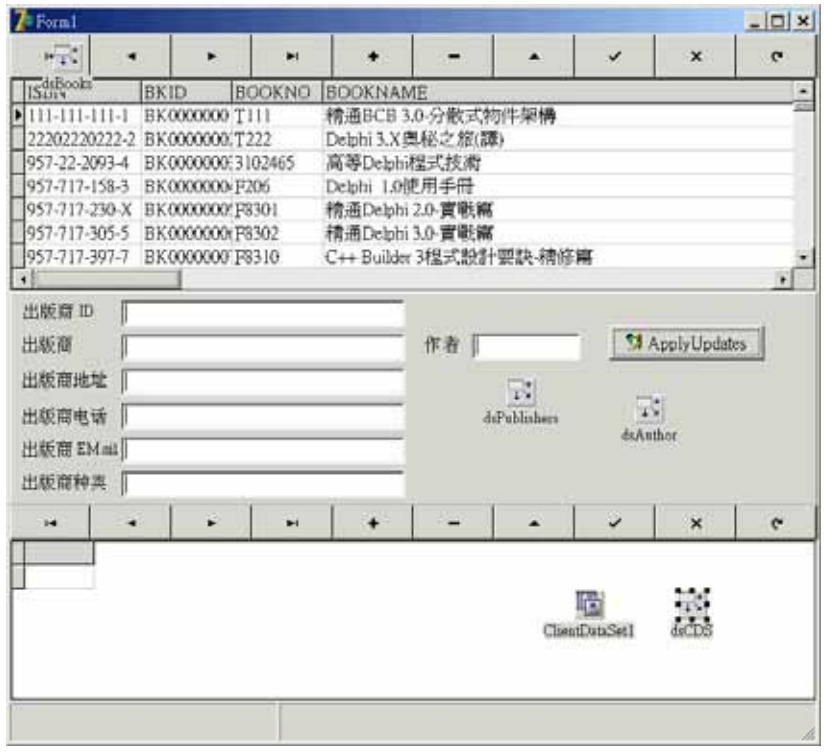


图2-12 在主窗体中加入TClientDataSet和其他可视化组件

现在，我们希望当用户修改了scdsBooks中的数据时把scdsBooks的Delta内容显示在刚才加入的TDBGrid中。因此我们可以在scdsBooks的AfterScroll事件中加入ShowDelta的过程调用代码：

```
procedure TdmDynamicSQL.scdsBooksAfterPost(DataSet: TDataSet);
begin
//  Self.scdsBooks.ApplyUpdates(0);
  Form1.ShowChangeCount;
  Form1.ShowDelta;
end;
```

而ShowDelta过程关键的地方则是把scdsBooks的Delta特性值指定给刚才加入的TClientDataSet组件的Data特性值。如此一来，连接到TClientDataSet的TDBGrid便能够显示用户修改了哪些数据。

```
procedure TForm1.ShowDelta;
begin
  cdsDelta.Data := dmDynamicSQL.scdsBooks.Delta;
end;
```

现在执行范例应用程序，并且试着修改scdsBooks中的任何数据，那么当你Post修改的数据之后，就会发现被修改的数据显示在主窗体下方的TDBGrid中。例如，图2-13是范例应用程序修改FXXX1这个记录的书名一栏的数值，在Post之后，请注意在下方的TDBGrid中显示了刚才笔者修改了BOOKNAME字段的数值。

在这里我们可以观察到Delta中的数值似乎是为每一次修改的数据保留两个记录。其中的第一个是未被更改的原始数据，而第二个则是被修改后的数据。但是在这个被修改后的记录中只有被修改的字段才会保存数值，其他没有被修改的字段则是空白的。这个现象就是DataSnap的运作原理，在稍后的章节中会详细说明DataSnap的运作方式。

既然我们已经能够通过使用TClientDataSet组件来巧妙地观察用户修改的数据，那么我们也可以使用相同的方式来观察TClientDataSet中当数据被修改后存储在Data中的特性值是否有任何的变化。

现在再回到主窗体中，再加入另外一组TClientDataSet、TDBNavigator和TDBGrid组件，就像刚才加入的一样。并且在主窗体的下方使用TPageControl组件，使用两个选项卡分别显示Data和Delta特性值。此时主窗体类似于图2-14。

接着再修改scdsBooks的AfterScroll事件程序代码，如下所示：

```
procedure TdmDynamicSQL.scdsBooksAfterPost(DataSet: TDataSet);
begin
//  Self.scdsBooks.ApplyUpdates(0);
  Form1.ShowChangeCount;
  Form1.ShowData;
```

```
  Form1.ShowDelta;
end;
```

上面的程序代码除了调用 ShowDelta之外，也调用了 ShowData来显示scdsBooks的Data特性值。而ShowData方法的实现程序代码则如下：

```
procedureTForm1.ShowData;
begin
  cdsData.Data := dmDynamicSQL.scdsBooks.Data;
end;
```

ShowData也和ShowDelta一样，只是它把scdsBooks的Data特性值指定给新加入的TClientDataSet组件。

现在再次执行范例应用程序，修改任何数据， Post修改的数据，那么就可以在下方的两个选项卡中看到 Data和Delta特性值。例如，图 2-15便是执行范例应用程序并且将原先的“实战 Delphi 7.x-Web Service专业程序设计”修改为“实战 Delphi 7.x-Web Service专业程序设计之二”后的画面。

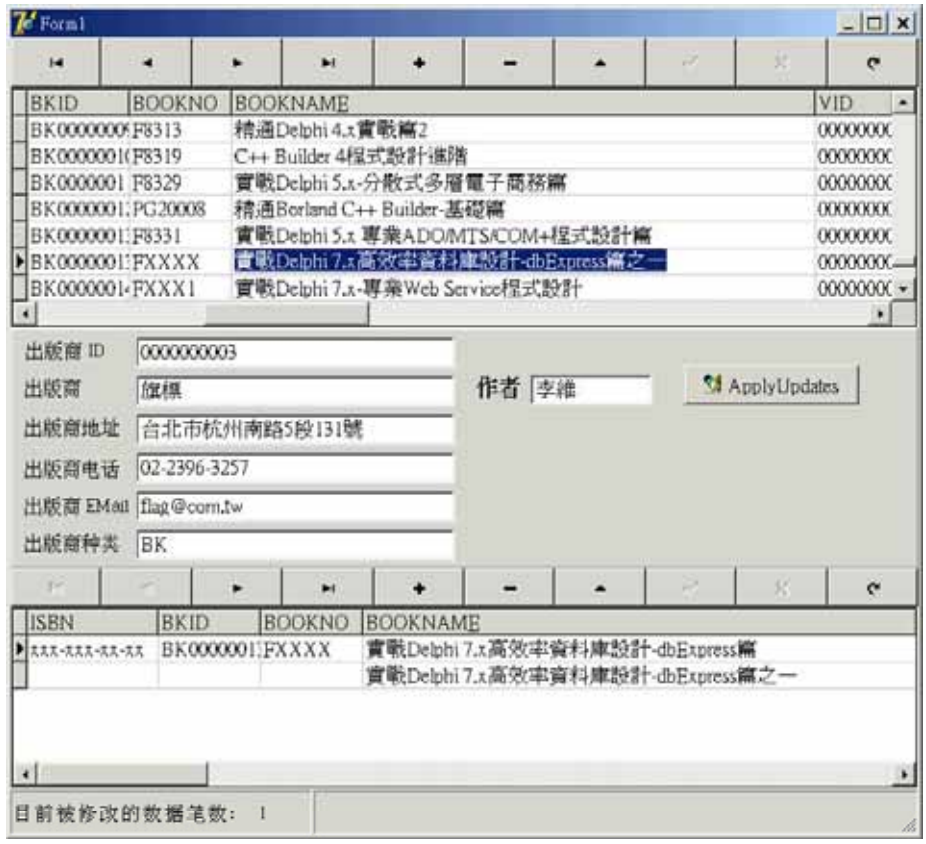


图2-13 范例应用程序显示了用户修改的数据

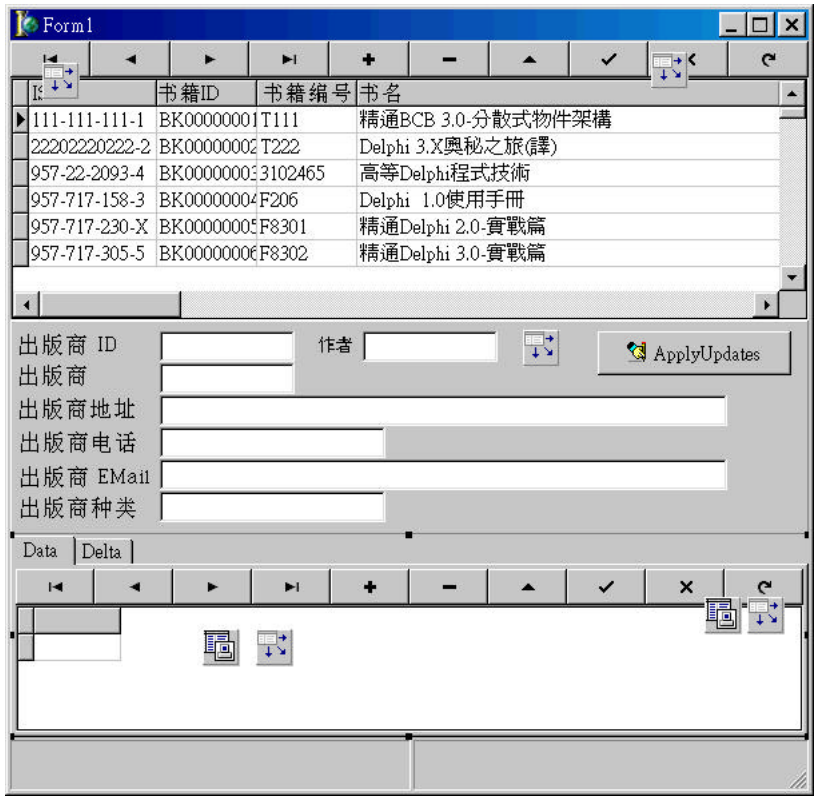


图2-14 在主窗体中再加入显示Data的数据感知组件

请注意，当Post修改的数据之后，原先存储在 scdsBooks的Data特性值中的原始数据也会跟着变成新的数值。

到此为止，本范例已经介绍了如何使用 dbExpress组件来访问数据并且把数据更新回数据源中。但是在这个范例的主窗体中， ApplyUpdates方法只能够把数据真正更新回BOOKS数据表中。如果我们希望用户能够同时更新 BOOKS和PUBLISHERS这两个数据表的数据，那么该如何做呢？下一小节会说明如何使用 dbExpress更新多个数据表的数据。

2.1.3 修改数据——多个数据表

在前面的范例应用程序中，主窗体同时显示了三个数据表的数据：书籍、出版商和作者。但是在前面 ApplyUpdates按钮的OnClick事件处理函数只调用了 scdsBooks的ApplyUpdates方法，因此即使用户同时修改了书籍和出版商的数据并且点击 ApplyUpdates按钮，那么也只有书籍的数据会更新回 BOOKS数据表中，出版商的数据并不会更新回 PUBLISHERS，因为范例应用程序并没有调用 scdsPublishers的

ApplyUpdates方法。那么范例应用程序如何同时将书籍和出版商的数据更新回这两个数据表呢？

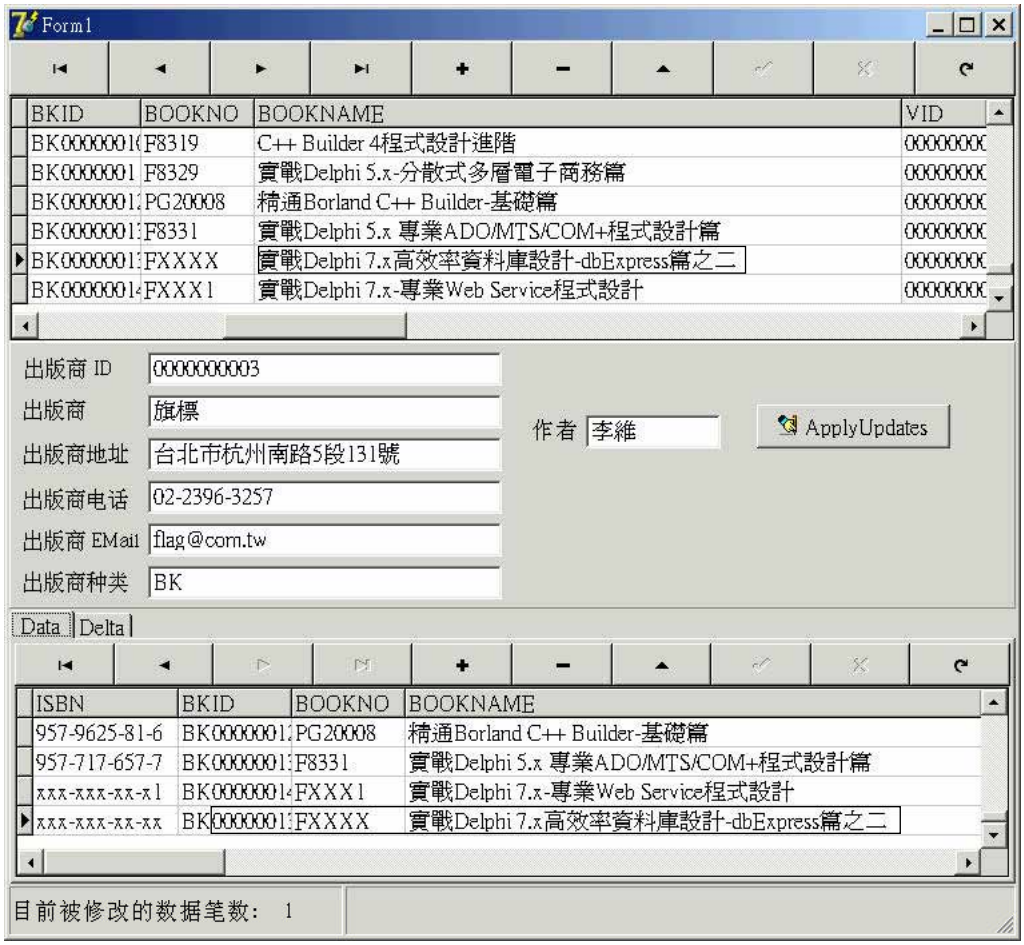


图2-15 范例应用程序显示scdsBooks的Data特性值内容

也许你会认为很简单，只要在ApplyUpdates按钮的OnClick事件处理函数中调用scdsPublishers的ApplyUpdates方法即可。但是，如果只是这么做的话，那么可能会发生问题，因为每一个ApplyUpdates方法会产生一个独立的数据库事务（Transaction），因此当scdsBooks和scdsPublishers分别调用ApplyUpdates方法时，便会激活两个独立的事务。因此如果用户同时更新了书籍和出版商的数据并且更新回数据表中，那么就有可能书籍数据更新成功，而出版商数据更新失败。因此如果我们希望在更新这两个数据表的数据时一定要同时成功，否则在任何一个数据表失败时就必须恢复到进行更新动作之前的状态，那么这两个更新操作就必须存在于同一个数据库事务中。数据库事务可以保证在同一个事务的范围中如果发生错误就把数据恢复

到原先的状态。

> 数据库事务会在第4章中详细说明。

在这个范例中我们希望同时更新数据，如果任何数据表发生错误，就必须恢复数据的状态。因此我们必须把这两个数据表更新动作包含在同一个数据库事务中。TSQLConnection组件的StartTransaction方法可以激活一个独立的数据库事务，因此我们可以先调用TSQLConnection的StartTransaction方法，再更新数据表的数据，如果数据表都成功地更新，就可以调用TSQLConnection的Commit方法保证同时更新成功。如果任何数据表发生错误，可以调用TSQLConnection的Rollback方法恢复数据。

TSQLConnection的StartTransaction方法原型如下：

```
procedure StartTransaction(TransDesc: TTransactionDesc);
```

它接受一个描述事务内容的参数TransDesc。TransDesc的类型是TTransactionDesc，在TTransactionDesc中用户可以指定特定的事务ID以及事务的级别。TSQLConnection之所以需要这个参数是因为dbExpress支持嵌套事务，这是比BDE先进的地方，在第4章中会继续讨论dbExpress的事务管理功能。

了解了如何把更新数据表的操作包含在事务范围中后，我们就可以修改主窗体中ApplyUpdates按钮的OnClick事件处理函数：

```
procedure TForm1.bbtnApplyClick(Sender: TObject);
var
  aTD: TTransactionDesc;
begin
if (not dmDemo1.scnnDemo.InTransaction) then
begin
    aTD.TransactionID := 1;
    aTD.IsolationLevel := xilREADCOMMITTED;
    dmDemo1.scnnDemo.StartTransaction(aTD);
    try
      if (dmDemo1.scdsBooks.ChangeCount)>0 then
        dmDemo1.scdsBooks.ApplyUpdates(0);
      if (dmDemo1.scdsPublishers.ChangeCount)>0 then
        dmDemo1.scdsPublishers.ApplyUpdates(0);
      dmDemo1.scnnDemo.Commit(aTD);
    except
      on e: Exception do
      begin
        ShowMessage(e.message);
        dmDemo1.scnnDemo.Rollback(aTD);
      end;
```

```
    end;
  end;
  ShowChangeCount;
end;
```

上面的程序代码首先通过TSQLConnection的InTransaction特性值来判断是否还有未结束的事务。如果没有的话，就设置一个 ID为1而事务级别为 xilREADCOMMITTED的TTransactionDesc对象，并且调用TSQLConnection的StartTransaction方法并且传入此对象。接着，程序判断 scdsBooks中是否有任何被修改过的数据，如果有的话，就调用它的ApplyUpdates方法。我们也同时对scdsPublishers进行一样的动作。最后，如果两个数据表都成功地更新，就调用 TSQLConnection的Commit方法确保更新操作的完成。当然如果发生任何错误，程序执行权便会转到 except部分。在这里程序代码先显示发生的错误消息，再调用 TSQLConnection的Rollback方法将数据恢复到进行更新操作之前的状态。由于 TTransactionDesc是在DBXpress程序单元中定义的，因此读者要记得在uses子句中加入DBXpress。

最后，再修改ShowChangeCount让它也在TStatusBar中显示用户对出版商修改的数据记录数。

```
procedure TForm1.ShowChangeCount;
begin
  StatusBar1.Panels[0].Text :=
    '目前被修改的数据笔数 : ' + IntToStr(dmDemo1.scdsBooks.ChangeCount);
  StatusBar1.Panels[1].Text :=
    '出版商被修改的数据笔数 : ' + IntToStr
  (dmDemo1.scdsPublishers.ChangeCount);
end;
```

现在再执行范例应用程序，并且同时修改书籍和出版商的数据。此时范例应用程序就应该看起来如图2-16所示。请注意下方的TStatusBar显示了这两个数据表目前被修改的记录数。此时，如果再点击主窗体中的 ApplyUpdates按钮，那么数据应该会成功地更新回这两个数据表中。如果观察 TSQLMonitor组件中的追踪消息，便可以看到如下的程序代码，这证明这两个更新操作是发生在同一个数据库事务中的。

```
INTERBASE - isc_start_transaction
INTERBASE - isc_dsql_allocate_statement
update BOOKS  set
 BOOKNAME = ?
where
 ISBN = ? and
 BKID = ? and
 BOOKNO = ? and
 BOOKNAME = ? and
 VID = ? and
```

```
  CATEGORY = ? and
  CLEVEL = ? and
  AID = ?
...
update PUBLISHERS  set
  ADDRESS = ?
where
  VID = ? and
  NAME = ? and
  ADDRESS = ? and
  PHONE = ? and
  EMAIL = ? and
  PKIND = ?
INTERBASE - isc_dsql_prepare
INTERBASE - isc_dsql_sql_info
INTERBASE - isc_vax_integer
INTERBASE - isc_dsql_describe_bind
INTERBASE - isc_dsql_execute
INTERBASE - isc_dsql_free_statement
INTERBASE - isc_dsql_free_statement
INTERBASE - isc_commit_retaining
```

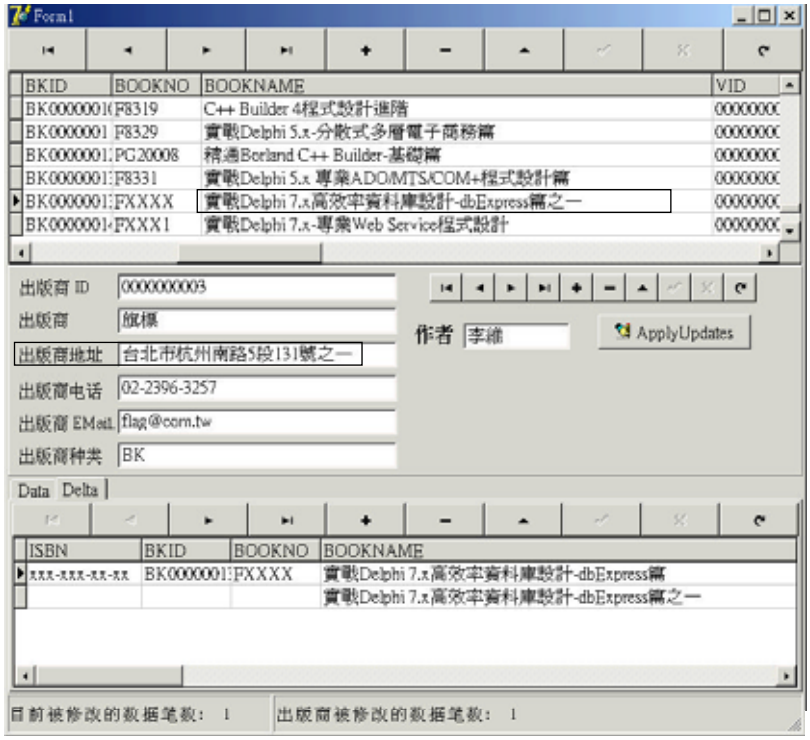


图2-16　在范例应用程序中同时修改BOOKS和PUBLISHERS这两个数据表的数据

现在的范例应用程序已经很完整了，它使用了多种经常用来开发数据库应用程序的技巧，也证明了前面讨论的内容。读者可以继续修改这个范例，例如如果用户也修改了作者信息，那么该如何把作者的数据更新回 PERFORMERS数据表呢？

2.1.4 控制数据访问记录数——PacketRecords特性

TSimpleDataSet的PacketRecords特性已经在前面介绍过了，这个特性对于整个dbExpress程序设计都有重大的影响，因此读者必须切实地了解并且掌握这个特性。PacketRecords的默认值是-1，这代表当TSimpleDataSet开启后，它会尽可能地把数据源中的数据一次读到客户端。对于拥有少量数据的数据源来说，这可能是很好的选择；但是对于拥有大量数据的数据源来说，这却是不好的设置。在一般的情形中，程序员可以将TSimpleDataSet的PacketRecords特性值调整为10到100之间，一次只访问应用程序需要的数据。这样可以让应用程序的反应时间良好，也可以节省客户端的资源。请记得一定要在真正需要一次访问所有数据时才设置 PacketRecords为-1。

现在让我们看一个小范例来说明 PacketRecords特性。首先在Delphi中建立一个新的应用程序，再建立一个新的数据模块，然后在数据模块中放入一个连接到范例数据库D7Books的TSQLConnection，放入一个TSimpleDataSet组件，设置它的 DataSet\CommandText为select * from PERFTEST。在PERFTEST数据表中拥有1000个记录，此时数据模块如图2-17所示。

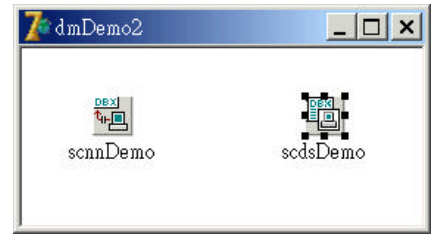


图2-17 范例应用程序的数据模块

在主窗体中加入 TDataSource、TDBNavigator、TDBGrid、TStatusBar以及一个TButton、TSpinEdit和一个TListBox组件（见图2-18）。将数据感知组件设置为连接到数据模块中的 scdsDemo。

在主窗体的“设置 PacketRecords”按钮的OnClick事件处理函数中，会根据用户在TSpinEdit中设置的数值来设置scdsDemo的PacketRecords特性值。下面就是这个按钮的OnClick事件程序代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  dmDemo.scdsDemo.Active := False;
  LogStartTime;
  dmDemo.scdsDemo.PacketRecords := sedtPacketRecords.Value;
  dmDemo.scdsDemo.Active := True;
  LogEndTime;
  LogMsg(sedtPacketRecords.Value, GetRunTime
end;
```

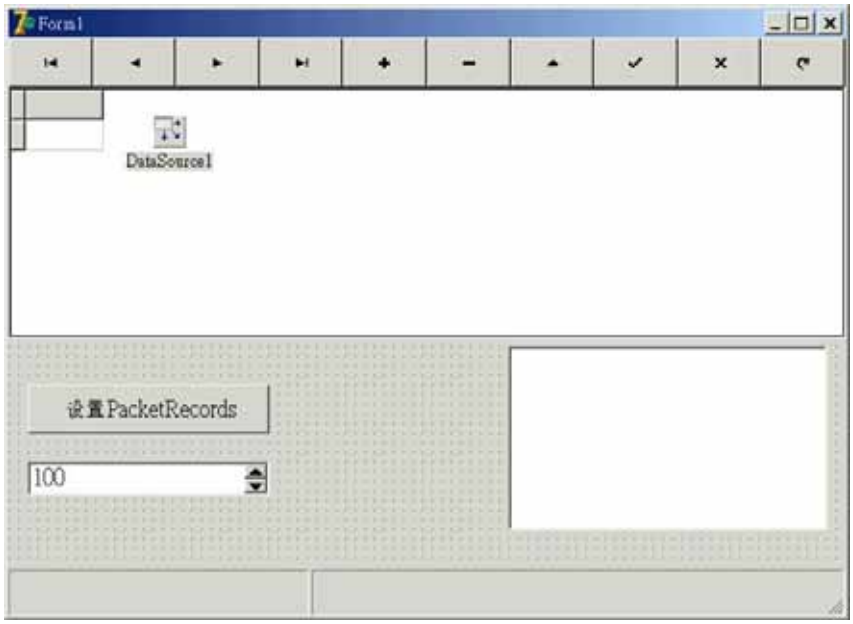


图2-18 范例应用程序的主窗体

上面的程序代码先关闭scdsDemo组件，再设置它的PacketRecords特性值，最后再开启scdsDemo，并且把这整个过程花费的时间显示在主窗体的TListBox中。

此外在范例应用程序执行时，我们也想要知道目前在客户端应用程序中dbExpress到底从数据源取得了多少个记录。为此，我们可以访问TSimpleDataSet组件的RecordCount特性值来得知目前在Data特性值中的记录数。

除了知道如何访问目前客户端的记录数之外，我们也必须决定什么时候比较适合访问记录数的信息。其中第一个时机便是当TSimpleDataSet被开启的时候，那就是它的AfterOpen事件。另外一个时机便是当TSimpleDataSet需要从数据源访问下一个数据封包时，这便是它的AfterGetRecords事件。因此我们只需要在这个两个事件中显示TSimpleDataSet的RecordCount特性值就可以知道目前在客户端被访问的记录数。下面的程序代码就是本范例应用程序在这两个地方显示RecordCount特性值的程序代码：

```
procedure TdmDemo.scdsDemoAfterOpen(DataSet: TDataSet);
begin
  Form1.ShowRecordCount(scdsDemo.RecordCount);
end;

procedure TdmDemo.scdsDemoAfterGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
```

```
  if (scdsDemo.Active) then
    Form1.ShowRecordCount (scdsDemo.RecordCount);
end;
```

现在编译并且执行范例应用程序。图 2-19便是将PacketRecords设置为10时开启TSimpleDataSet的画面。在主窗体下方的TStausBar3中可以看到目前客户端果然只访问了10个记录。

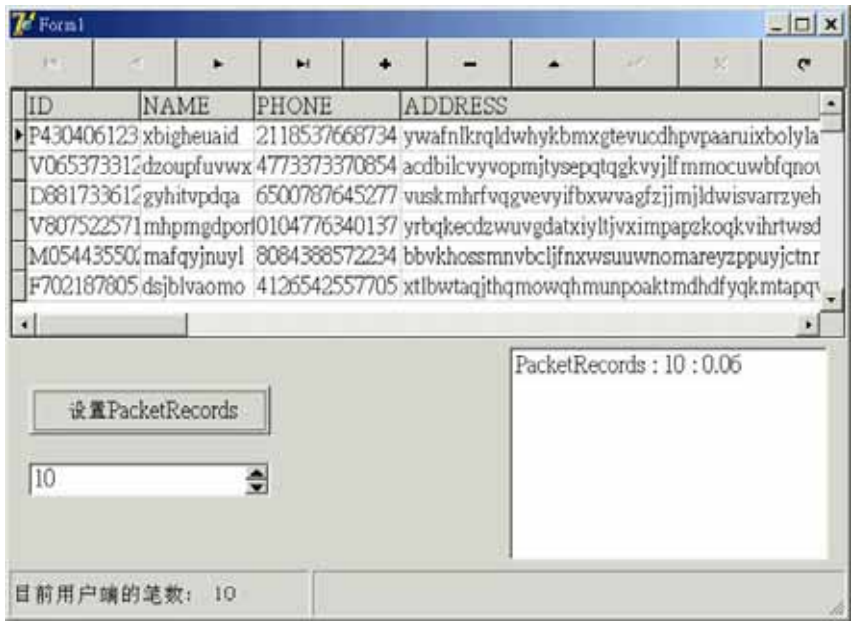


图2-19 将PacketRecords设置为10时范例应用程序的执行画面

现在，如果我们点击 TDBNavigator不断地向下浏览数据，便会发现当浏览第 11个记录时，TSimpleDataSet会再从数据源自动取得下一个数据封包，此时 TStatusBar也会显示记录数已经成为 20了（见图 2-20）。所以证明了 TSimpleDataSet每次以PacketRecords指定的数目从数据源读取数据。

图2-21则是在PacketRecords为100时从数据源读取数据，不管 PacketRecords值为何，TSimpleDataSet的执行行为都是一样的，但是我们发现范例应用程序使用 100作为PacketRecords值似乎比10运行得快，读者可以自行试着调整 PacketRecords值，看看不同的PacketRecords值会有什么执行结果。

这个范例显示了TSimpleDataSet的PacketRecords可以控制一次从数据源中取得的记录数，让程序员能够控制客户端应用程序需要的数据。另外，我们也发现不同的PacketRecords特性值会使应用程序产生不同的反应时间，程序员可以根据自己的情况来设置PacketRecords特性值，一般来说如果 PacketRecords特性值不是设置成 - 1，那么10到1000之间是比较好的设置。当然如果使用客户端的人数多，那么程序员可

能需要将PacketRecords特性值减少为10到100之间。

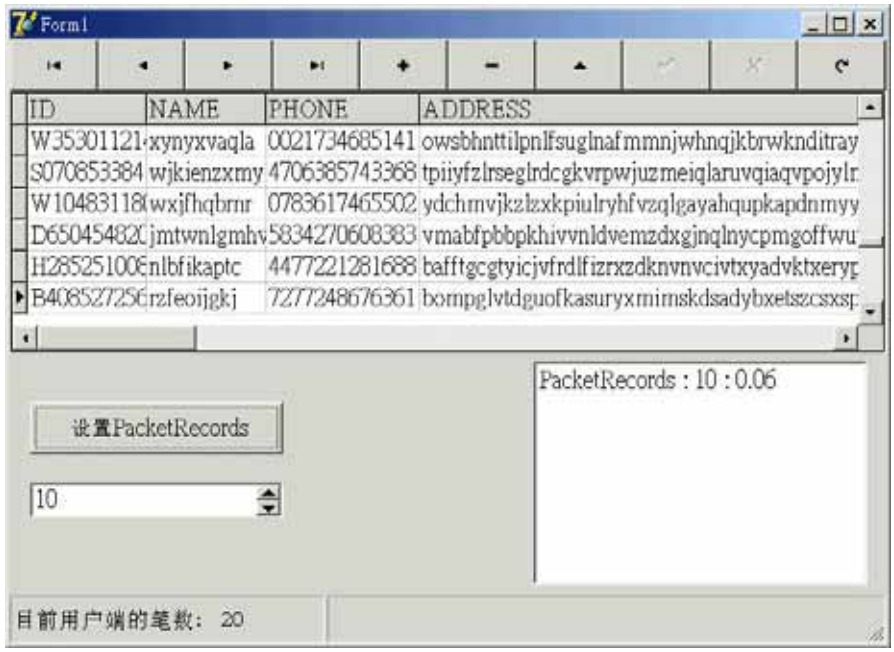


图2-20　范例应用程序会自动读取下一个数据封包

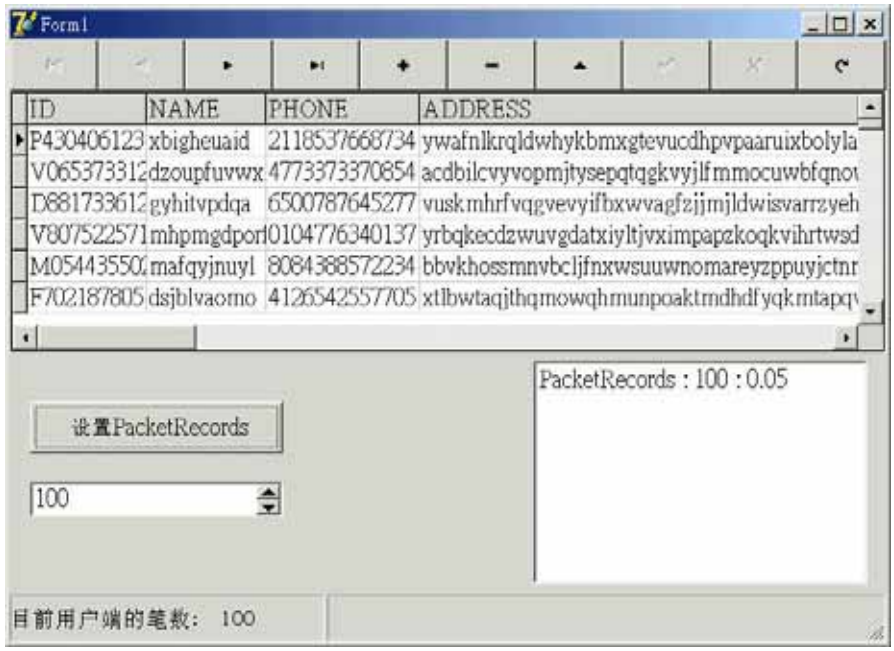


图2-21　范例应用程序使用100作为PacketRecords值时的执行画面

这本高效率数据库程序设计书其实在 Delphi 6时就已经开始编写了，不过笔者到了Delphi 7才推出本书。在笔者把 Delphi 6的程序更改成 Delphi 7时发现Delphi 7的dbExpress的性能又比Delphi 6时又好了很多，许多执行时间比 Delphi 6时又快了10%~30%，实在令人非常惊讶且兴奋，看来 Borland仍然在不断地改善dbExpress的性能，也许在读者阅读本书时 dbExpress又有了进步。

2.2 DataSnap技术

Delphi 7的dbExpress是一组组件和数据访问引擎，虽然程序员在开发数据库应用程序时是使用dbExpress，但是在dbExpress的内部却使用了DataSnap技术。Delphi 7的DataSnap也就是以前Delphi中的MIDAS技术，只是Delphi 7强化了MIDAS的功能，还让MIDAS能够同时在Windows和Linux平台上执行。

DataSnap是以数据封包（Data Packet）的方式来传递数据的，当客户端应用程序使用dbExpress访问数据时，在dbExpress的内部会使用SQL语句从数据源取得结果数据集，然后把结果数据集中的数据转换为数据封包，再存储到 TSimpleDataSet或是TClientDataSet的Data特性值中。而数据封包则是以 OleVariant的类型代表的，因此TSimpleDataSet的Data和Delta特性值都是OleVariant类型的数值。

而当应用程序修改了数据时，这些修改的数据便存储在 TSimpleDataSet的Delta特性值中，并且当应用程序调用 ApplyUpdates方法更新数据时，Delta特性值便会传递到dbExpress的内部，再由DataSnap根据这些修改的数据自动转换为 SQL语句，再执行这些SQL语句把数据更新回数据源中。图2-22便显示了这个处理流程。

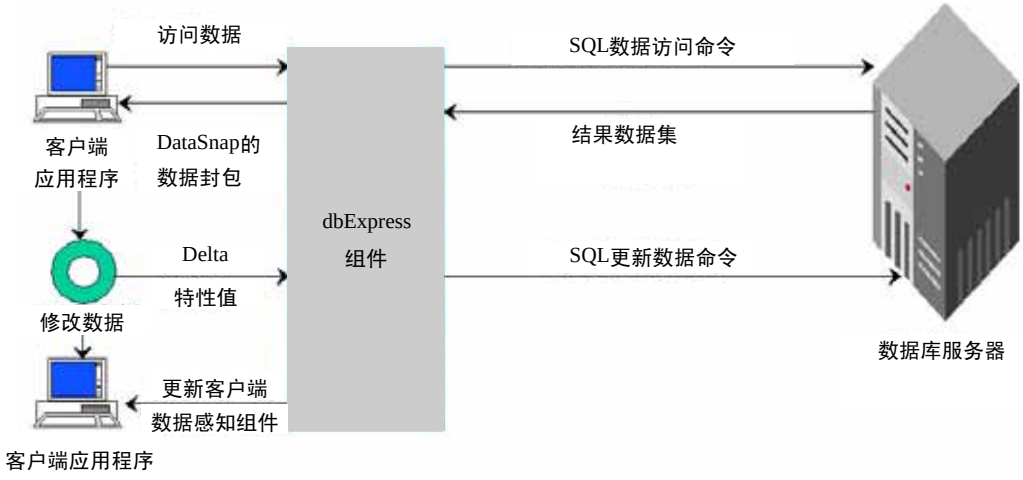


图2-22 dbExpress客户端应用程序如何从数据源取得数据和处理数据

如果客户端应用程序以分段的方式访问数据，即设置 TSimpleDataSet的Packet-Records特性值为一个正数，那么当用户在客户端浏览或是访问的数据不在目前的Data特性值中时，dbExpress和DataSnap便会自动地从后端数据源中访问下一个数据封包。TSimpleDataSet组件会自动调用 GetNextPacket访问下一个数据封包，一直到所有的数据都已经读入TSimpleDataSet的Data特性值为止，图2-23说明了以分段访问数据的流程。

让TSimpleDataSet以分段的方式访问数据的好处是可以降低应用程序的反应时间，也可以在多人使用时降低网络的负荷。这对于多人使用的系统而言是比较好的数据访问模式。

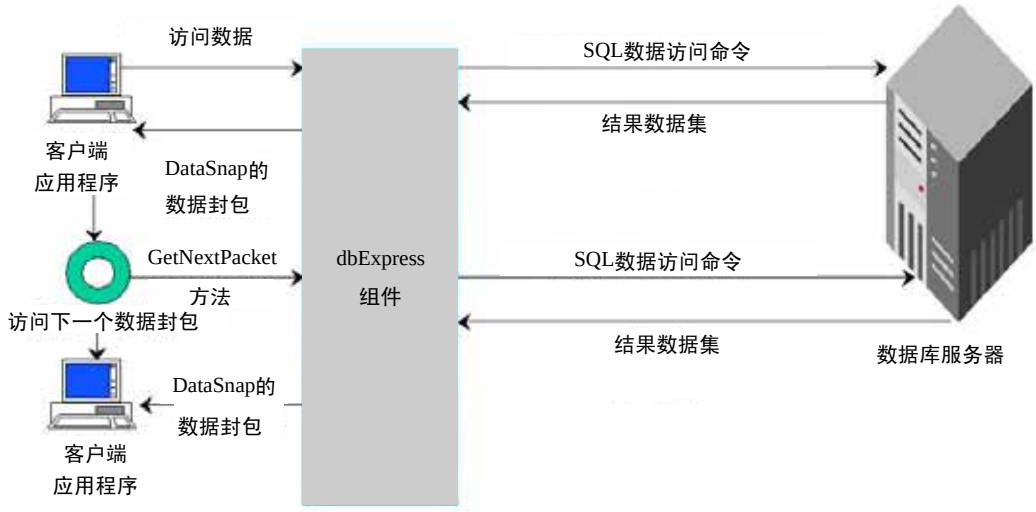


图2-23 dbExpress应用程序如何使用分段的方式访问数据

当TSimpleDataSet以分段的方式访问数据时，它也会触发 TSimpleDataSet组件中相应的事件处理函数。例如，在 GetNextPacket真正访问数据之前，它会触发TSimpleDataSet的BeforeGetRecords事件；在访问下一数据封包之后，它又会触发AfterGetRecords事件。刚才在前面的范例中我们也是利用 AfterGetRecords事件来显示目前在客户端的记录数。图2-24显示了分段访问数据的流程以及相应的事件。

在2.1.2小节中，本书讨论了 Data和Delta特性值，并且通过范例应用程序观察到当用户修改数据之后 Data和Delta特性值的变化。现在让我们以一个场景范例来说明在DataSnap中对于修改的数据的处理流程。

图2-25说明了客户端应用程序使用 dbExpress和DataSnap传递数据以及修改数据的流程。一开始，客户端应用程序要求数据源传递 100个记录，当 dbExpress和DataSnap收到要求后，就从数据源取得这 100个记录，封装成数据封包并且传递给客户端应用程序。

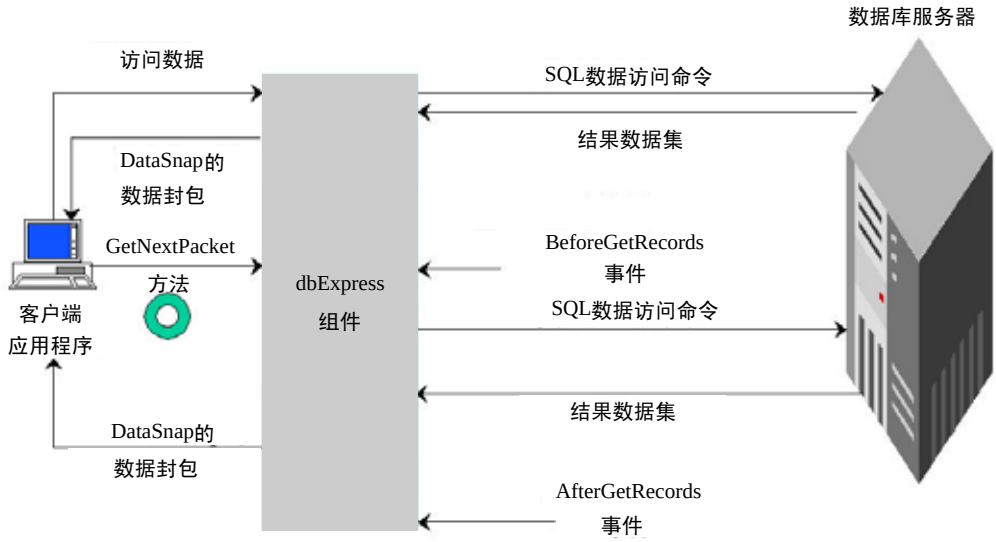


图2-24 分段访问数据时的过程以及触发的事件

客户端的用户在程序中修改数据，例如用户改变了客户端中 Joe的记录，把Joe的职位从Accountant改成Senior Accountant，然后调用ApplyUpdates把这个记录封装成Delta数据封包返回给dbExpress和DataSnap，要求更新客户端对于Joe这个记录的修改。

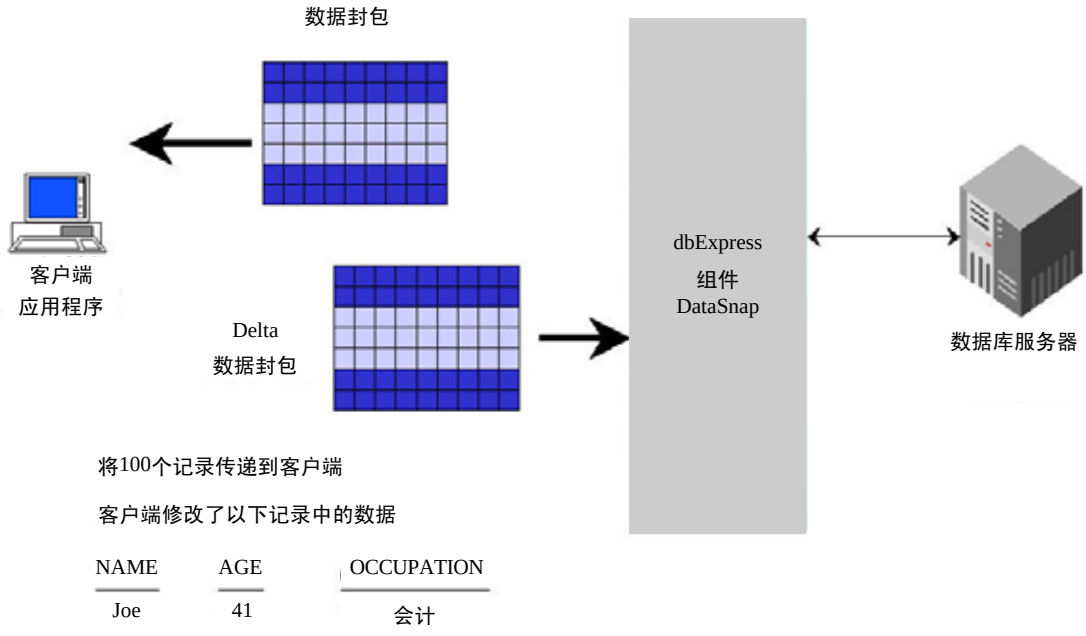


图2-25 多层应用系统传递数据的实例

当客户端封装修改数据的数据封包时，它是按照图 2-26显示的方式来封装修改数据的。首先DataSnap会把未修改之前的数据封装在数据封包中，然后再封装修改后的数据。请注意在封装的第二个记录中只有被改变过的字段才会有数值，此外在每一个记录中还会有一个额外的字段来说明这个记录的修改方式，例如在这个例子中Joe这个记录是被修改的，所以它的 OCCUPATION字段包含Senior Accountant值，UPDATE TYPE字段会拥有Modify值。

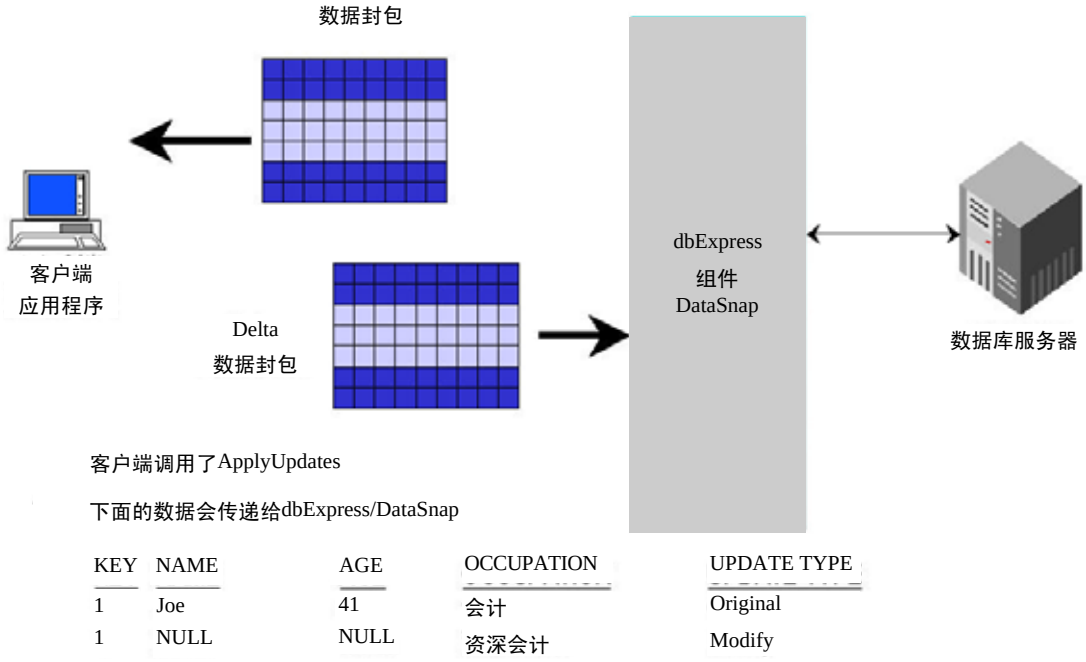


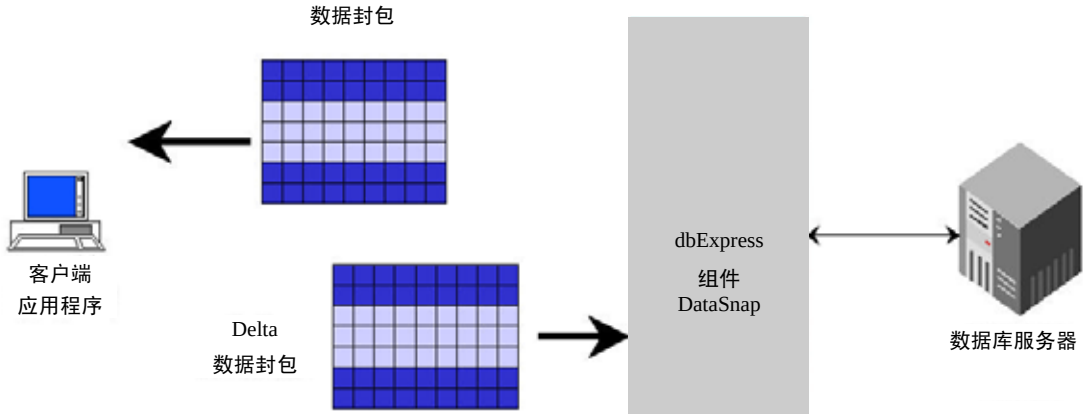
图2-26 客户端传递修改过的数据向应用程序服务器请求更新

从上面的说明中可以知道，对于在客户端修改的每一个记录而言，当这些数据要传递回数据源进行更新时，dbExpress和DataSnap会封装两个记录。DataSnap对于封装的第二个记录只记录被修改的字段，这样可以节省网络通信资源，加快传递和执行的效率。

在刚才的范例中，如果DataSnap发现要被更新的记录已经被其他用户改变了，例如Joe这个记录的AGE字段已经被改成42，与原先的41不一样，此时DataSnap便会把客户端原来传送来的数据，再加上数据库中最新的这个记录传送回客户端，要求客户端应用程序或是用户决定如何处理。图 2-27是这个数据处理流程的示意图。

当更新错误的数据到达客户端之后，客户端应用程序可以在 OnReconcileError事件处理函数中对应用程序服务器返回的造成错误的数据进行处理。当然，客户端应用程序也可以把这些数据呈现给用户，让用户决定如何处理这些数据（见图 2-28），

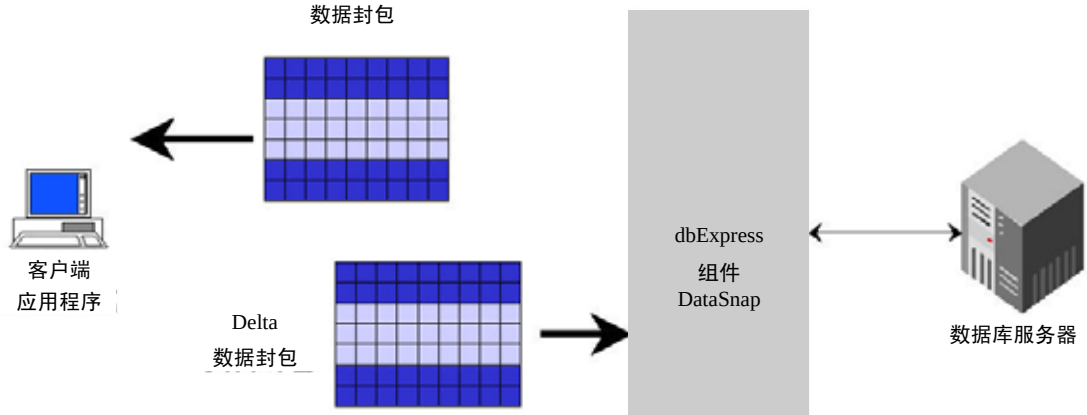
在稍后的章节中会说明如何处理dbExpress/DataSnap更新数据发生的错误。



如果要修改的数据已经被其他用户改变了，
那么就会返回错误
返回的错误可能包含以下信息：

KEY	NAME	AGE	OCCUPATION	UPDATE_TYPE
1	Joe	41	会计	Original
1	NULL	NULL	资深会计	Modify
1	Joe	42	会计	New Values

图2-27 应用程序服务器更新数据发生错误时的处理流程



程序员可以决定如何处理错误
在复杂的情况下，程序员可以将错误和
造成错误的数据返回给用户，并且要求
用户采取行动

图2-28 修改数据的错误

本小节解释了DataSnap的基本概念和处理数据的过程。虽然在一般的dbExpress应用程序中程序员可能并不需要直接使用DataSnap技术，因为这些工作都由dbExpress组件组自动帮助程序员解决了。但是在许多高级的应用中使用DataSnap的概念和技术却是必要的，也可以帮助程序员解决许多困难和调整应用程序的性能。在本书稍后的章节中，读者将会逐渐接触到DataSnap技术，并且了解如何使用它发挥dbExpress更大的效能。

2.3 使用TSQLDataSet和TSQLQuery组件

dbExpress的TSQLDataSet和TSQLQuery组件提供了几乎一样的功能，只是TSQLQuery比较类似于BDE中的TQuery组件，对于熟悉BDE的程序员来说是比较容易上手的。而TSQLDataSet则是一般通用的数据访问组件，可以在许多的场合使用。

一般来说，TSQLDataSet和TSQLQuery组件都是使用SQL语句来处理数据的。由于通过这两个组件访问的数据并不能让用户修改，因此它们通常都是和TClientDataSet以及TDataSetProvider组件一起使用，以便让用户可以处理和修改数据。不过对于一些属于DDL（Data Definition Language）的SQL语句来说，这两个组件就非常适合了。例如，建立数据库中的数据表、删除数据表或是增加索引等的工作就很适合使用这两个组件来进行。

要使用TSQLDataSet组件，程序员可以设置它的CommandType特性来指定执行的目标是SQL语句或是存储过程，就如同前面小节介绍TSimpleDataSet的CommandType特性一样。接着在TSQLDataSet的CommandText特性中下达SQL语句或是选择存储过程名称，最后再设置它的Active特性值或是调用ExecSQL方法。

使用TSQLQuery组件和使用TSQLDataSet几乎一样，只是TSQLQuery组件只执行SQL语句，而且设置SQL语句的地方是在它的SQL特性中。简单地说，TSQLQuery就等于一个将CommandType设置为ctQuery的TSQLDataSet组件。

现在就让我们使用实际的范例来说明如何使用这两个组件。

2.3.1 使用TSQLDataSet组件

本小节的范例将会展示如何使用TSQLDataSet组件来执行DDL语句，在InterBase数据库中动态建立数据表、建立主键并且删除数据表。

步骤1：建立数据模块和dbExpress组件

在Delphi集成开发环境中点击File|New|Application建立一个新的Delphi/Kylix应用程序。接着点击File|New|Data Module建立空白的数据模块。在此数据模块中加入TSQLConnection组件，连接到范例数据库D7Books，接着放入3个TSQLDataSet组件，

一个命名为sdsRunDDL，另外一个命名为sdsDropTable，最后一个命名为sdsCreateIndex。再放入一个TSimpleDataSet组件，命名为scdsQuery并且设置它的CommandText特性值为select * from Books，此时数据模块看起来如图2-29所示。

图2-29 范例应用程序的数据模块

在sdsRunDDL组件的CommandText特性值中加入如下的SQL语句。这个SQL语句会在数据库中建立一个名为MYESSAYS的数据表。

```
CREATE TABLE MYESSAYS(
    EID INTEGER NOT NULL,
    ETITLE VARCHAR(60),
    MAGAZINE VARCHAR(60),
    PDATE DATE,
    CONTENTS BLOB sub_type 0 segment size 80,
    NOTES VARCHAR(100));
```

在sdsCreateIndex组件的CommandText特性值中加入如下的SQL语句。这个SQL语句会在MYESSAYS数据表中建立一个以EID字段为基础的主键。

```
ALTER TABLE MYESSAYS ADD  PRIMARY KEY(EID);
```

最后一个sdsDropTable组件则是执行DROP Table SQL语句以删除由sdsRunDDL动态建立的数据表。

```
DROP TABLE MYESSAYS
```

步骤 2：建立范例主窗体

回到范例应用程序的主窗体，在其中放入TDataSource并且将它连接到数据模块中的scdsQuery，再放入TDBNavigator和TDBGrid。最后放入两个TButton组件，一个设置Caption特性值为“建立数据表”，另外一个设置Caption特性值为“删除数据表”。此时主窗体如图2-30所示。

步骤 3：实现范例应用程序

双击主窗体中的“建立数据表”按钮，并且在它的OnClick事件处理函数中编写如下的程序代码：

```
dmRunDDL.sdsRunDDL.ExecSQL (False) ;
dmRunDDL.sdsCreateIndex.ExecSQL (False) ;
dmRunDDL.scdsQuery.Active := False;
dmRunDDL.scdsQuery.CommandText := 'Select * from MyEssays';
dmRunDDL.scdsQuery.Active := True;
```

在上面的程序代码中，执行了sdsRunDDL组件中的SQL语句。由于sdsRunDDL组件的SQL语句是不返回结果数据集的 Create Table语句，因此我们必须调用TSQLDataSet的ExecSQL方法，而不是Open方法。

图2-30 范例应用程序的主窗体

ExecSQL方法的原型如下：

```
function ExecSQL (ExecDirect: Boolean = False):Integer;override;
```

ExecSQL方法用来执行不返回结果数据集的SQL语句或是属于DDL的SQL语句。它接受一个ExecDirect参数，这个参数代表TSQLDataSet在执行SQL语句之前是否需要先准备（编译）这个SQL语句。如果ExecDirect是False就代表要先准备SQL语句，如果ExecDirect是True则代表直接执行SQL语句而不需要先准备。如果程序员需要执行相同的SQL语句数次，那么传递False给ExecSQL方法是比较好的。

当sdsRunDDL执行完毕之后，MyEssays数据表应该就被建立了，因此上面的程序代码再指定Select * from MyEssays给scdsQuery，并且开启MyEssays数据表。如果sdsRunDDL正确而且成功地执行，那么在范例应用程序中的TDBGrid中应该可以看到MyEssays这个数据表了。

接着双击主窗体中的“删除数据表”按钮，并且在它的OnClick事件处理函数中编写如下的程序代码：

```
dmRunDDL.sdsDropTable.ExecSQL (False) ;
```

上面的程序代码调用了sdsDropTable的ExecSQL方法执行存储在sdsDropTable组件中的SQL语句，以删除MyEssays数据表。

步骤 4：执行范例应用程序

现在可以编译并且执行范例应用程序。图 2-31中的画面就是执行范例应用程序的情形，在一开始时范例应用程序显示了 Books数据表中的数据。当我们点击了“建立数据表”按钮之后，范例应用程序会动态建立 MyEssays数据表，并且将它显示在TDBGrid中。

图2-31 执行范例应用程序的画面

现在，如果到数据库管理工具中检查，便可以看到类似图 2-32的画面，证明了范例应用程序果然在数据库中成功地动态建立了 MyEssays数据表。

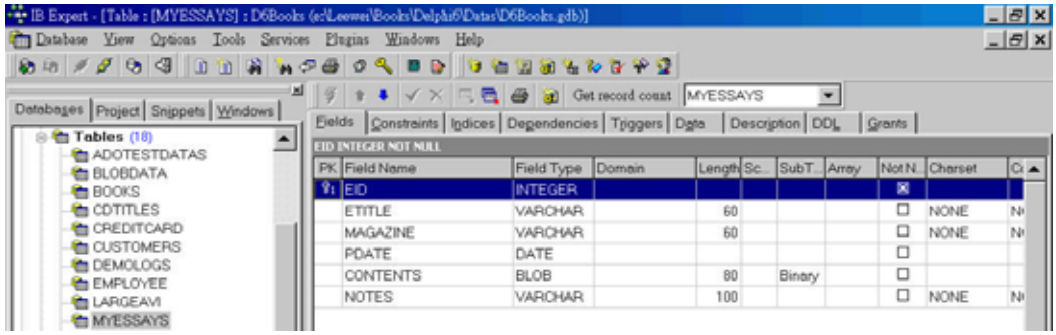


图2-32 范例应用程序在InterBase中建立了MyEssays数据表

这个范例应用程序展示了如何使用 TSQLDataSet动态建立数据表，在随后的小节中会继续讨论如何使用TSQLQuery组件在动态建立的数据表中新增数据。

2.3.2 使用TSQLQuery组件

在上一小节中演示了如何使用 TSQLDataSet动态建立数据表，现在让我们继续使用TSQLQuery来展示如何在动态建立的数据表中新增数据。

步骤 5：加入TSQLQuery组件

打开范例的数据模块，并且在其中加入一个 TSQLQuery组件，命名为 sqlqData，如图2-33所示。

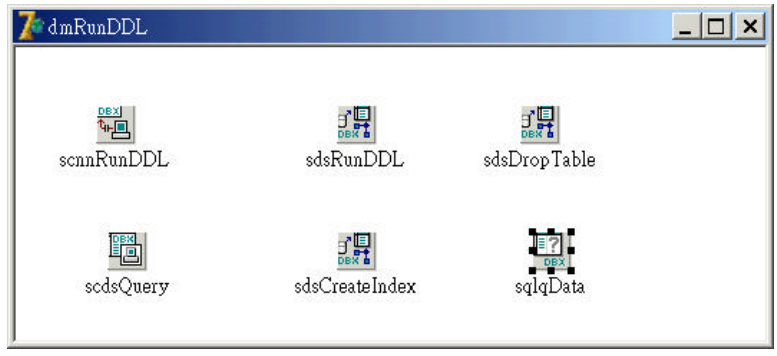


图2-33 在数据模块中加入TSQLQuery组件sqlqData

设置它的SQLConnection特性值为数据模块中的 scnnRunDDL，并且将它的SQL特性值设置为：

```
INSERT INTO MYESSAYS(EID, ETITLE, MAGAZINE, PDATE, NOTES) VALUES (1, '200年软
件巨星-Kylix', 'RUN!PC 2001/3', '03/05/2001 00:00:00', NULL
```

这个SQL语句会在MYESSAYS数据表中新增一个新记录。

步骤 6：使用TSQLQuery组件新增数据

回到范例应用程序的主窗体，加入一个新的 TButton组件，设置它的Caption特性值为“新增数据”，双击这个TButton组件，并且在它的OnClick事件处理函数中编写如下的程序代码：

```
procedure TForm1.btnInsertDataClick(Sender: TObject);
begin
  dmRunDDL.sqlqData.ExecSQL(False);
  dmRunDDL.scdsQuery.Refresh;
end;
```

上面的程序代码调用了 sqlqData的ExecSQL方法，让它执行存储在 sqlqData组件中的SQL语句，最后再调用数据模块中的 TSimpleDataSet组件的Refresh方法从数据表中得到最新的数据。

> 注意到了吗？以前 BDE的TQuery在许多的情形下都无法调用 Refresh方法重新取得数据，而dbExpress的TSQLDataSet、TSQLQuery和TSQLClientDataSet组件却都可以自由地调用Refresh，这比BDE好用得多。

现在再次执行范例应用程序，点击“建立数据表”按钮，再点击“新增数据”按钮，那么便会看到类似图 2-34的画面，TSQLDataSet和TSQLQuery组件都可以成功地执行DDL语句。

图2-34 范例应用程序动态创建数据表以及新增数据的画面