第六章　用 dbExpress 访问

数据表和查询数据库

6.1　访问数据表

访问数据表需要通过至少两个控件：一个数据库连接控件—TSQLConnection 和一个数据集控件—TSQLDataSet/TSQLQuery/TSQLTable。关于 TSQLConnection 控件，在第四章已作过说明，本节主要介绍各种数据集控件的使用。

6.1.1　使用 TSQLDataSet/TSQLQuery/TSQLTable 控件的一般步骤

TSQLDataSet、TSQLQuery 和 TSQLTable 是三个可以用于访问数据表的 dbExpress 数据集控件，它们和 Delphi 所提供的其它数据连接方式的数据集控件十分相似，如：TSQLDataSet 跟 TADODataSet、TIBDataSet 用法相似，而 TSQLQuery、TSQLTable 则跟 TQuery、TTable、TADOQuery、TADOTable、TIBQuery、TIBTable 等类似。但它们必须通过 TSQLConnection 连接数据库后，才能连接到数据表，而 ADO/BDE 的数据集控件都是可以不通过 TADOConnection/TDatabase 直接连接数据表的（虽然不推荐这样做），这一点有较大不同，反而是与 InterbaseExpress 类似。

dbExpress 的应用结构一般如图 1：

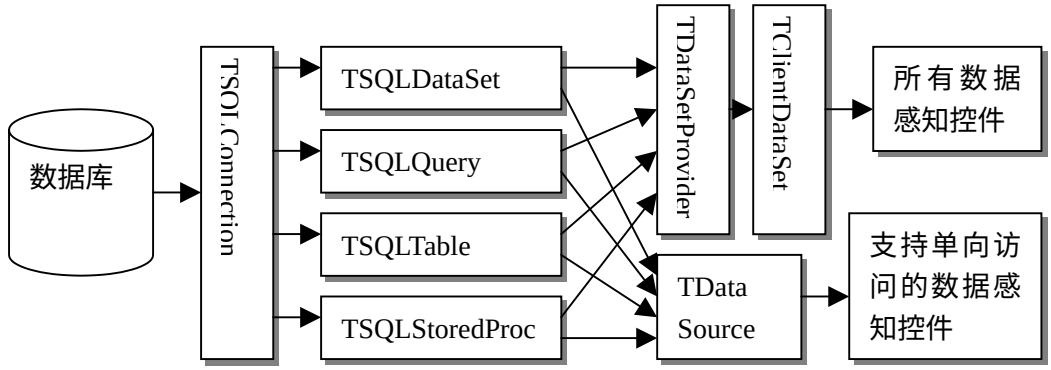

图 1：dbExpress 的一般应用结构

图 2 中，右上部为双向数据访问方式结构，右下部为单向数据访问方式的结构，在单向数据访问结构的末端只能使用支持单向访问的数据感知控件，如 TDBEdit 等，而不能使用 TDBNavigator 和 TDBGrid 等需要双向数据访问支持的数据感知控件。

这三个数据集控件与其它连接方式的数据集控件一样，都是从 TDataSet 类

派生出来的，其派生关系如图 2：

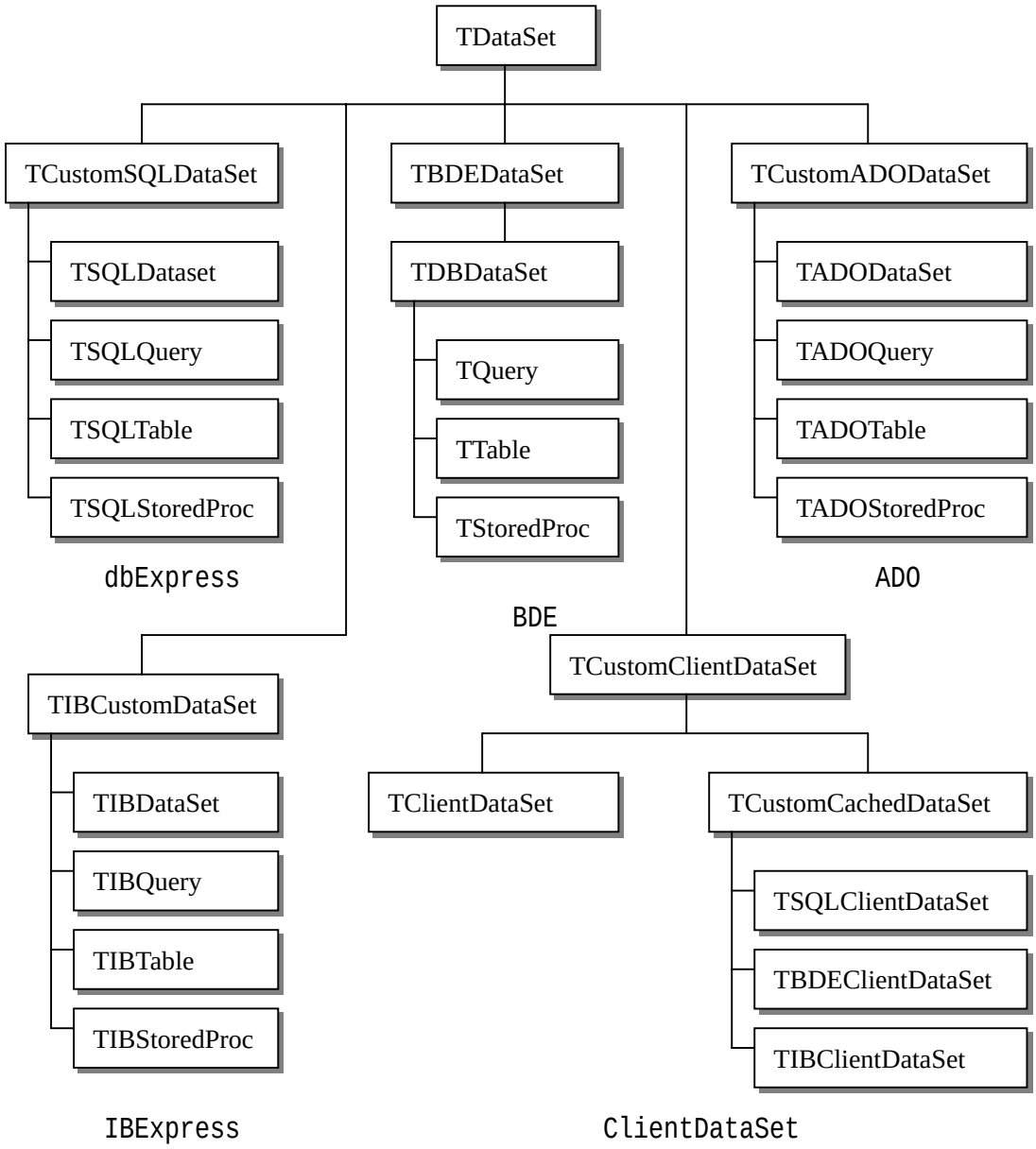

图 2：各种数据集控件的派生关系

因为 TSQLDataSet、TSQLQuery、TSQLTable 都是从 TDataSet 中派生而来，所以它们都拥有和其它数据集控件几乎相同的大多数属性、方法和事件。这些方法可以分为几大类：数据集的打开/关闭、数据浏览、数据编辑、数据过滤、书签管理等。由于 dbExpress 的数据集控件都是只支持单向游标，所以相应的属性、方法、事件较其它数据集控件要少很多。

下面主要以 TSQLDataSet 为例作介绍，TSQLQuery 和 TSQLTable 与此类似。习惯了 BDE 编程的读者可能会对 TSQLDataSet 控件比较陌生，不过熟悉 ADO 编程的读者可能就会比较了解了，因为 ADO 中也有一个类似的 TADODataSet。其实 DataSet 控件和 Query/Table 控件很相似，但相比之下要灵活一些，它即可以像 Query 类控件那样通过 SQL 语句进行复杂的多表查询，也可以像 Table 类控件那样以表为单位进行数据访问，它甚至还可以像 StoredProc 类控件那样操作存储

过程。至于效率方面则与其它控件并无不同。读者在编程中可以按自己的习惯选择合适的数据集控件，习惯用 BDE 的可能选 TSQLTable 和 TSQLQuery，习惯用 ADO 的可能选 TSQLDataSet，本书主要以 TSQLDataSet 为例，TSQLTable 和 TSQLQuery 请参照使用。

1. TSQLDataSet 的打开和关闭：

TSQLDataSet 可以以三种方式打开：通过 SQL 语句以 Query 方式打开，通过数据表名以 Table 方式打开，通过存储过程名以 StoredProc 方式打开。但不论以哪种方式打开，在打开之前都必须将 TSQLDataSet 连接到一个 TSQLConnection 控件，这一点与 BDE/ADO 不同。

- 以 Query 方式打开：

  将 TSQLDataSet 的 CommandType 属性设为 ctQuery，然后在 CommandText 属性中设置 SQL 语句。
- 以 Table 方式打开：

  将 TSQLDataSet 的 CommandType 属性设为 ctTable，然后在 CommandText 属性中输入数据表名。
- 以 StoredProc 方式打开：

  将 TSQLDataSet 的 CommandType 属性设为 ctStoredProc，然后在 CommandText 属性中输入存储过程名。

按上述设置时要注意先修改 CommandType，再修改 CommandText，因为 CommandType 改变时会自动将 CommandText 清空。

设置好上面的两个属性后，就可以打开数据集，也有两种方法：

- 将 Active 属性设为 True；
- 调用 Open 方法；

这与其它数据集控件是一样的，而且两种方法的效果也是完全相同。数据集只有在打开状态下才能进行浏览等操作。

数据集的关闭很简单，用与打开类似的两种方法：

- 将 Active 属性设为 False；
- 调用 Close 方法；

2. TSQLDataSet 的浏览：

与一般数据集组件相同，TSQLDataSet 也有一组用于数据浏览的方法：

- First 将游标移动到第一条记录，即将第一条记录设为当前记录；
- Last 将游标移动到最后一条记录；
- Next 移动到下一条记录；
- Prior 移动到前一条记录；
- MoveBy 移动若干条记录，参数对相对记录数，正数表示向后移，负数向前。

虽然 TSQLDataSet 提供了这些函数，但是因为它是单向游标的数据集，所以有一些限制，主要是反向移动游标的限制。首先，Last 方法是不能执行的，在 TSQLDataSet（包括 TSQLQuery 和 TSQLTable）中执行此方法将导致一个异常，表示此方法不可用于单向数据集；其次，Prior 方法也是不能执行的；最后 MoveBy 的参数不可以为负数，因为这样也表示向后移动游标，这也是不支持的。

因为这个原因，所以不可以将 TDBNavigator 用于此类数据集控件，因为它

不能完全支持 TDBNavigator 的所有功能。

但 Eof 和 Bof 两个属性还是可以用的。Eof 为 Ture 时表示当前记录为最后一条；Bof 为 True 时表示当前记录为第一条。

3. TSQLDataSet 的编辑：

因为 TSQLDataSet 没有像 TQuery 那样的 RequestLive 属性，用于实现对简单查询的数据更新，也没有像 TADODataSet 那样能够自动更新的功能，也不支持像 TIBDataSet 那样集成三个更新语句（Insert/Update/Delete）来实现更新，所以 TSQLDataSet 只能是一个只读数据集，虽然它也提供了以下方法：

- Edit　　使数据集进入编辑状态；
- Insert　　插入一条记录；
- Append　追加一条记录；
- Post　　更新数据；

但是在完成数据更新调用 Post 方法提交时会导致一个异常，表示此数据集为只读，不可更新。

要对 TSQLDataSet 进行数据更新只有两个办法：

- 将 CommandType 设为 ctQuery，然后在 CommandText 中填入更新数据的 SQL 语句，如：INSERT/UPDATE/DELETE，然后调用 ExecSQL 方法执行，即可实现数据更新（包括插入/修改/删除），在后面的单向数据访问方式介绍中将介绍用此方法进行数据更新；
- 用后面将要详细介绍的 Provider/Resolve 机制，TDataSetProvider 将自动实现数据更新操作；

4. TSQLDataSet 的过滤：

同样是因为单向数据访问的限制，TSQLDataSet 不支持 Filter（过滤器），虽然 Filter 和 Filtered 两个属性在 Object Inspector 中看不到，是因为是 Public 的属性，而不是 Published，可以在程序中访问，但是无效。TSQLQuery 和 TSQLTable 也是这样。

如果要用 TSQLDataSet 进行数据过滤，一个办法是在 CommandText 属性的 SQL 语句里加上 WHERE 子句，在其中加上过滤条件；另一个方法是用 Provider/Resolve 机制，在 TClientDataSet 中使用 Filter。

5. TSQLDataSet 的书签：

也是因为单向数据访问方式的限制，TSQLDataSet 不支持书签，虽然包括 TSQLDataSet 在内的 dbExpress 数据集控件都提供了全套的书签管理方法：

- GetBookmark　　在数据集的当前位置设置一个书签
- BookmarkValid　　检查书签是否有效
- CompareBookmarks　比较两个书签
- GotoBookmark　　移动到书签指定的位置
- FreeBookmark　　释放书签

但是由于 GotoBookmark 的参数（即书签）所指定的位置可能在当前位置之前，所以 dbExpress 的 GetBookmark 根本就不能放书签，其返回值永远是一个非法的书签，即调用 BookmarkValid 验证结果总为 False。

如果要使用书签功能，用 Provider/Resolve 是唯一的方法。

6.1.2 单向数据访问

如前面所述，单向的数据访问方式的功能非常有限，除了可以进行部分数据浏览功能外，其它的如数据编辑和数据过滤只能用变通的方法实现，而书签功能则根本无法实现。

接下来将以一个例子来全面地介绍一下单向数据访问的方式。

程序的 Form 如图 3：

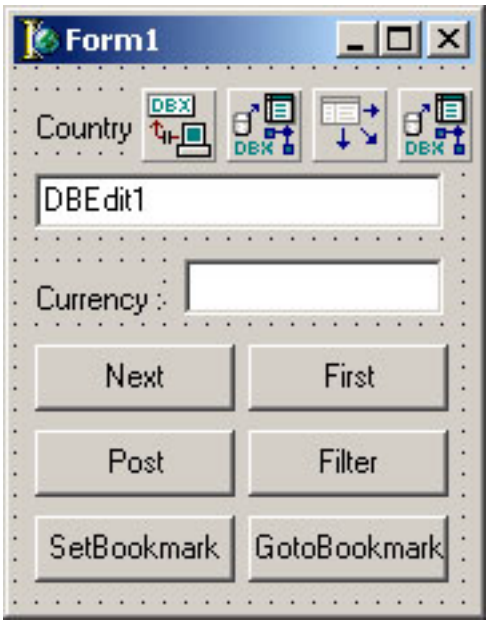

图 3：单向数据访问方式的例子 Form

其中各控件的属性设置如表 1：

表 1：单向访问方式的例子 Form 的控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径 >\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
SQLDataSet1	CommandText	select COUNTRY, CURRENCY from COUNTRY
	SQLConnection	SQLConnection1
DataSource1	DataSet	SQLDataSet1
DBEdit1	DataSource	DataSource1

	DataField	COUNTRY
SQLDataSet2	SQLConnection	SQLConnection1
其它控件	Caption	如图 3

最后按后面的程序清单设置事件响应，并写入程序代码。完整的程序清单如下：

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, DBXpress, StdCtrls, DB, DBCtrls, FMTBcd, SqlExpr, Mask;

type
  TForm1 = class(TForm)
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Button2: TButton;
    DataSource1: TDataSource;
    DBEdit1: TDBEdit;
    SQLDataSet2: TSQLDataSet;
    Label2: TLabel;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
  private
    { Private declarations }
    FBookmark : Pointer;
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    FBookmark := Nil;
    SQLConnection1.Open;
    SQLDataSet1.Open;
    Edit1.Text := SQLDataSet1.FieldByName( 'CURRENCY' ).AsString;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    SQLDataSet1.Next;
    Edit1.Text := SQLDataSet1.FieldByName( 'CURRENCY' ).AsString;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    SQLDataSet1.First;
    Edit1.Text := SQLDataSet1.FieldByName( 'CURRENCY' ).AsString;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    SQLDataSet2.Close;
    SQLDataSet2.CommandText := 'UPDATE COUNTRY SET CURRENCY = '''
        + Edit1.Text + ''' WHERE COUNTRY = '''
        + SQLDataSet1.FieldByName( 'COUNTRY' ).AsString + ''' ';
    SQLDataSet2.ExecSQL( );
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
    SQLDataSet1.Close;
    SQLDataSet1.CommandText := 'select COUNTRY, CURRENCY from COUNTRY
WHERE COUNTRY LIKE ''%a%'' ';
    SQLDataSet1.Open;
    Edit1.Text := SQLDataSet1.FieldByName( 'CURRENCY' ).AsString;
```

```
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
    If ( ( FBookmark <> Nil ) AND SQLDataSet1.BookmarkValid( FBookmark ) )
Then
        SQLDataSet1.FreeBookmark( FBookmark );
    FBookmark := SQLDataSet1.GetBookmark;
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
    If ( ( FBookmark <> Nil ) AND SQLDataSet1.BookmarkValid( FBookmark ) )
Then
        SQLDataSet1.GotoBookmark( FBookmark );
end;

end.
```

对上面的程序有必要作一个说明：

- 数据集的打开

  如程序中 Form1 的 OnCreate 事件的响应，即 TForm1.FormCreate 方法中所示。首先将书签记录变量 FBookmark 设为空，然后连接到数据库并打开数据集，在打开数据集后，将其第一条记录的 CURRENCY 字段显示在 Edit1 控件中；由于 DBEdit1 是一个数据感知控件，它会同时显示 COUNTRY 字段的内容。

- 数据集的浏览

  如程序中 Button1 和 Button2 的事件响应。

  Button1 表示移动到下一条记录，它调用了 Next 方法实现，同时它还会将新的 CURRENCY 字段内容取出并用 Edit1 显示。

  Button2 表示移动到第一条记录，它通过调用 First 方法来实现，它也会更新 Edit1 的显示。

  当然 DBEdit1 的显示是自动更新的。

- 数据集的编辑

  如程序中 Button3 的事件响应。Button3 使用了另一个 TSQLDataSet 控件（SQLDataSet2）来进行数据更新操作。

  浏览数据时，在 Edit1 中输入新的 CURRENCY 值，再按 Button3（Post 按钮），即可将数据更新到数据库中，按 First 再一步步 Next 到那一条记录，即可看到 CURRENCY 字段已被修改。

  其方法是根据 Edit1 和 SQLDataSet1 的当前记录（以 COUNTRY 字段为主键）来生成一条数据更新的 SQL 语句（UPDATE 语句），并运行此语句来实现此功能的。也可以对此作一些修改来实现对 DBEdit1 修改的数据更新。

- 数据集的过滤

  如程序中 Button4 的响应。按 Button4 按钮，将把数据集中所有 COUNTRY

字段中包含有小写字母’a’的记录过滤出来。

这是通过关闭数据集后，修改 CommandText 属性，将其 SQL 语句加上 WHERE 子句的过滤条件，再打开此数据集实现。按过此按钮后再一步步 Next 即可看到只有 COUNTRY 字段中含有小写字母’a’的记录才会显示，其它记录都被过滤掉了。

注意：TSQLDataSet 之类的单向访问数据集来说，Filter 和 Filtered 属性虽然也有，但是不可用，如果在程序中设置它们将导致一个异常，说明此功能不被支持。

- 书签管理

程序中 Button5 和 Button6 实现书签管理的功能。Button5 用于设置书签，Button6 用于移动到书签所设的位置，但实际使用中，这个功能没有效果，不过也不会像 Filter 之类那样出错。因为 dbExpress 虽然提供了这些书签管理函数，但没有实现，主要是 GetBookmark 不能设置书签，且返回值是一个无效的书签，所以不能用。

要实现此功能，只能使用下面要说的 Provider/Resolve 机制。

6.1.3 双向数据访问

正因为单向数据访问方式的局限，为了完全实现数据库访问的功能，必须使用 Delphi 6 的 DataSnap（即以前的 MIDAS）中的 TDataSetProvider 和 TClientDataSet 两个控件来实现 Provider/Resolve 机制。

这两个控件在程序中的作用如图 4：

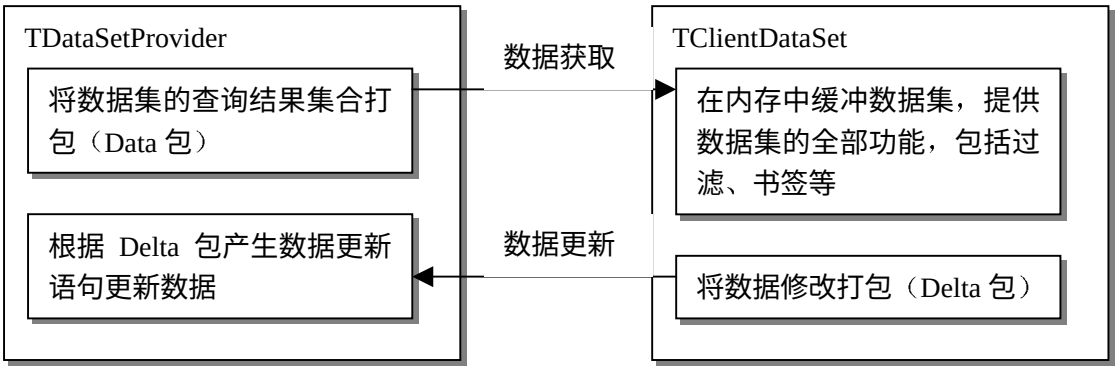

图 4：Provider/Resolve 的功能

以一个例子来说明双向数据访问的实现，程序的 Form 如图 5：

图 5：双向数据访问方式的例子 Form

其中各控件的属性设置如表 2：

表 2：双向访问方式的例子 Form 的控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径>\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
	Connected	True（前提是先设置好上面的属性）

SQLDataSet1	CommandText	select COUNTRY, CURRENCY from COUNTRY
	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet1	ProviderName	DataSetProvider1
	Active	True（前提是先设置好上面的属性）
DataSource1	DataSet	ClientDataSet1
DBNavigator1	DataSource	DataSource1
DBGrid1	DataSource	DataSource1
其它控件	Caption	如图 5

最后按后面的程序清单设置事件响应，并写入程序代码。完整的程序清单如下：

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, Grids, DBGrids, ExtCtrls, DBCtrls, DBXpress, FMTBcd, DB,
  DBClient, Provider, SqlExpr, StdCtrls;

type
  TForm1 = class(TForm)
    DBNavigator1: TDBNavigator;
    DBGrid1: TDBGrid;
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
    DataSetProvider1: TDataSetProvider;
    ClientDataSet1: TClientDataSet;
    DataSource1: TDataSource;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private
    { Private declarations }
```

```
    FBookmark : Pointer;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    FBookmark := Nil;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    ClientDataSet1.ApplyUpdates( -1 );
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    ClientDataSet1.Filter   := 'COUNTRY LIKE ''%a%'' ';
    ClientDataSet1.Filtered := True;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    If      (      (      FBookmark      <>      Nil      )      AND
ClientDataSet1.BookmarkValid( FBookmark ) ) Then
        ClientDataSet1.FreeBookmark( FBookmark );
    FBookmark := ClientDataSet1.GetBookmark;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
    If      (      (      FBookmark      <>      Nil      )      AND
ClientDataSet1.BookmarkValid( FBookmark ) ) Then
        ClientDataSet1.GotoBookmark( FBookmark );
end;

end.
```

对上面的程序说明如下：

首先，通过 ClientDataSet 对数据集的缓冲，提供了全部数据集操作功能，所以这里可以使用 TDBNavigator 和 TDBGrid 这两个必须有双向数据访问功能支持的数据感知控件。

- 数据集的打开

  因为数据集在设计期间就设置为打开状态，即 SQLConnection1 的 Connected 属性为 True，并且 SQLDataSet/ClientDataSet 的 Active 属性为 True，这样当程序运行时就会自动打开数据集。

  当窗体创建完成，即 OnCreate 事件响应中，将书签记录变量设置为空。

  由于 DBGrid1 是一个数据感知控件，它会在窗体显示时，同时显示出数据集的全部内容。

- 数据集的浏览

  通过 DBNavigator1 的 First record、Prior record、Next record、Last record 四个按钮可以方便而且自由地移动到任何一条记录上。

  也可以在 DBGrid1 中通过键盘或鼠标操作，进行随意的移动。

- 数据集的编辑

  通过 DBNavigator1 的 Insert record、Delete record、Edit record 三个按钮可以方便地增加、删除和编辑记录数据，然后通过 Post edit 按钮将数据的修改提交，或者按 Cancel edit 放弃此次修改，不过即使此时提交修改也只是将数据的修改提交到 ClientDataSet 的缓冲数据集中，要将数据更新提交到数据库，必须按 Apply 按钮。

  在 Button1（Apply）按钮的响应中调用了 ClientDataSet1 的 ApplyUpdates 方法，它有一个参数，表示在数据更新提交过程中出现多少错误时中止数据提交，在这个例子中，这个参数为-1，表示不管出现多少错误，都不停止提交，直到数据更新的提交全部完成；如果此参数为 0，则表示出现任何错误都将中止数据提交过程。更详细的关于数据更新提交的内容将在后面讨论。

  同样在 DBGrid1 中也可以通过键盘进行数据的插入、删除和编辑，当从编辑过的记录中移动离开时将自动把修改 Post 到 ClientDataSet1 中。在 DBGrid1 中所作的修改同样要按 Apply 按钮才能真正更新到数据库中。

- 数据集的过滤

  按 Button2（Filter）按钮可以实现数据过滤，它将把数据集中所有 COUNTRY 字段中包含有小写字母'a'的记录过滤出来。

  这种过滤与前面单向数据访问方式的过滤方法不同，这里只是对缓冲数据集中的记录进行过滤，这种过滤不需要重新打开和关闭数据集。它是通过将 ClientDataSet1 的 Filter 属性设置上过滤条件，并将 Filtered 属性设为 True，使过滤条件有效即可。要取消数据过滤，只需要将 ClientDataSet1 的 Filter 属性清空，或将 Filtered 属性设为 False 即可。

- 书签管理

  将当前数据记录移动到任何位置，按 Button3（SetBookmark）按钮即可将此记录设置为书签记录，然后再移动到其它位置，按 Button4（GotoBookmark）即可将当前记录移动到书签所设置的位置。

  这里的实现方法与单向访问方式是一样的，只是操作的数据集改为

ClientDataSet1 的缓冲数据集，因为这个缓冲数据集提供了双向访问能力，所以支持书签功能。

6.1.4 数据更新

从前面双向访问方式的例子中可以看出，它的数据更新方式与其它数据集控件的数据更新方式很不相同，因为它多了一级缓冲，所以 Post 操作只是将数据修改更新到 ClientDataSet 的缓冲数据集中，必须增加一个 ApplyUpdates 的操作才能真正把修改更新到实际的数据库中。

其实为 MIDAS/DataSnap 提供的 Provider/Resolve 机制的数据更新不仅仅只是增加 ApplyUpdates 一个步骤那么简单，它的功能要强大得多。图 6 是 Provider/Resolve 的数据更新过程：

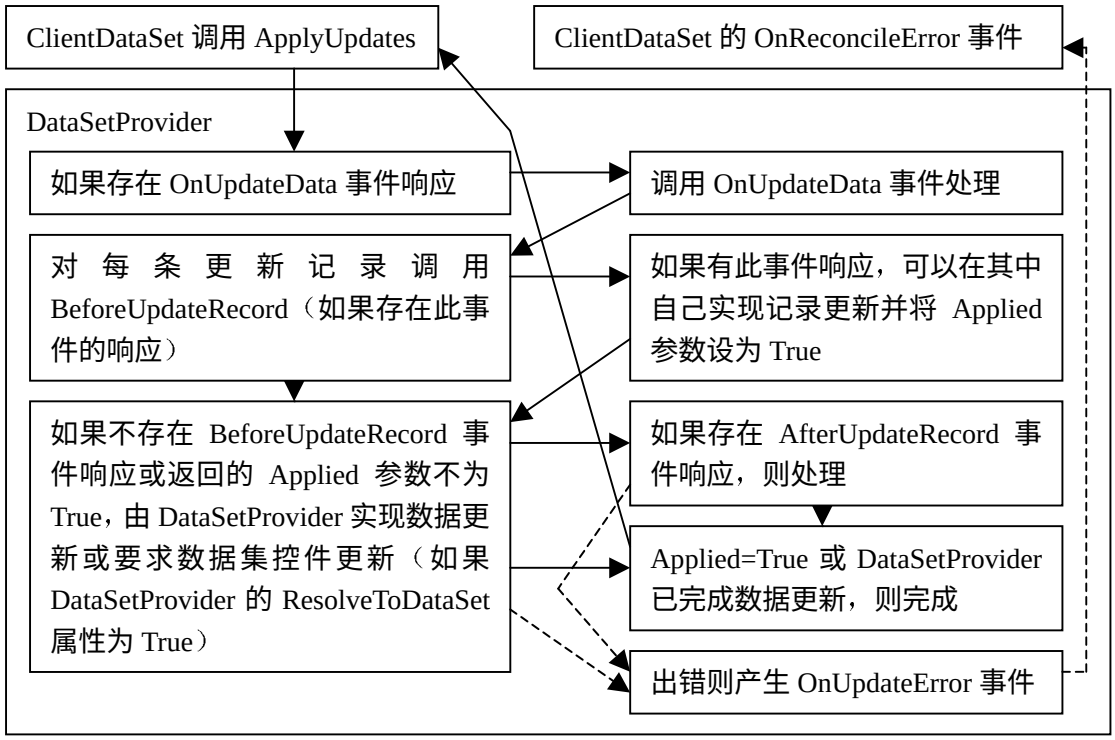

图 6：Provider/Resolve 机制的数据更新过程

从图中可以看出，我们可以在数据更新的几个地方进行自定义数据更新操作。

- 首先，可以在 DataSetProvider 的 OnUpdateData 事件中处理成批的数据更新操作。
- 其次，可以在 DataSetProvider 的 BeforeUpdateRecord 事件中处理记录的更新操作。

  当然，如果只是想在记录更新之前作一些处理，而仍然由 DataSetProvider 进行默认的数据更新操作，只需在此事件中将 Applied 参数设为 False 即可。

  否则，如果是自己已经将记录更新完成，就要将此参数设为 True。

对于 DataSetProvider 的默认数据更新也有两种方式：

- 当 DataSetProvider 的 ResolveToDataSet 属性为 False 时，由 DataSetProvider 产生数据更新语句进行数据更新。

  这种方式适用于 BDE、IBExpress、dbExpress 等数据库连接方式，当然 ADO 也可以，不过不如另一种方法好。
- 当 DataSetProvider 的 ResolveToDataSet 属性为 True 时，数据的更新操作由数据集控件产生。

  这种方式适用于 ADO 连接方式，因为 ADO 能够自动产生数据更新语句，并且自动支持多表查询的更新，功能比 DataSetProvider 更强大。

当数据更新发生错误时，将先在 DataSetProvider 中产生 OnUpdateError 事件。可以在此事件响应中检查出错的原因，如果可能，还可以在此事件中纠正错误，通过 Response 参数，可以告诉 DataSetProvider 如何对待此错误，如忽略此错误或是跳过此记录等。

如果 ClientDataSet 调用 ApplyUpdates 时用的参数不为 0，则会在 ClientDataset 中产生 OnReconcileError 事件，同样可以在此事件对错误进行处理。如果 ApplyUpdates 的参数为 0，则在发生错误时将停止更新操作。

在 ClientDataSet 调用 ApplyUpdates 提交更新操作前可以通过 ClientDataSet 的 ChangeCount 属性来了解已经有多少条记录被修改并 Post 了，也就是将要被 Apply 的记录数。

6.1.5 在数据表中搜索记录

对于单向访问方式来说，唯一的记录搜索方法就是通过关闭数据集，在 CommandText 属性中写入 SQL 语句再打开数据集来实现。但这种方法只适用于结果记录集的搜索，而不能定位到所要的记录上，因为要搜索的记录可能在当前位置之前，所以 Locate 方法也像 Filter 一样，虽然存在，但不被支持，如果在程序中对单向数据集控件使用 Locate 方法，在运行时会导致异常。

同样，为了解决这个问题，也要采用 Provider/Resolve 机制。在 TClientDataSet 的缓冲数据集中，可以使用 Locate 或 Lookup 方法来搜索特定的记录。

Locate 用于在数据集中搜索特定的记录，并将当前记录设置为此记录。Locate 方法有三个参数：KeyFields、KeyValues 和 Options。

- KeyFields　是一个字符串，记录要查找的字段，可以有多个，只要将字段名用分号“;”隔开即可；
- KeyValues　是一个 Variant 型的变量，通常用 Variant 数组记录搜索条件，即要求的相应字段的值；
- Options　　是搜索选项，是一个可以有两个选项的集合类型，这两个类型为可选的搜索附加条件：
  - loCaseInsensitive　表示用大小写无关方式查找；
  - loParialKey　　表示不必完全匹配，即模糊查找；

下面是一个用 Locate 搜索记录的例子：

```
If ( ClientDataSet1.Locate( ‘Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
```

```
'831-431-1000']), [loPartialKey]) ) Then
    ShowMessage( 'Found it!' );
```

上面这个例子搜索 Company 字段为 Sight Diver，Contact 字段为 P，Phone 字段为 831-431-1000 的记录，并且用模糊查找但大小写必须相同。

如果找到了符合条件的记录，就将当前记录设置为此条记录并返回 True，否则返回 False。如果找到多条符合条件的记录，则定位到第一条符合条件的记录上。

当然，在很多情况下不需要搜索这么多条件，如果只要求一个字段匹配就会很简单：

```
Locate( 'Company', 'Sight Diver', [] );
```

这个例子表示查找 Company 字段为 Sight Diver，并且是精确查找。

Lookup 和 Locate 很相似，不过它不是将当前记录定位到搜索到的记录上，而是返回搜索到的记录中的某些字段的值，并且它不会改变当前记录的位置。Lookup 方法也有三个参数：KeyFields、KeyValues 和 ResultFields，其中前两个参数跟 Locate 方法是一模一样的，最后一个 ResultFields 表示要取得搜索结果记录的哪些字段，也可以有多个字段，以分号分隔。

Lookup 方法的返回值为搜索到的记录的相应字段值，因为可以有多个字段，并且类型不一定，所以 Lookup 的返回值是一个 Variant 类型。如果结果为多个字段，则 Lookup 返回一个 Variant 数组。下面是一个例子：

```
Var
    v : Variant;
Begin
    v := ClientDataSet1.Lookup( 'Company', 'Sight Diver', 'Contact;Phone' );
    If ( VarIsArray( v ) ) Then
        ShowMessage( String( v[0] ) + ' ' + String( v[1] ) );
End;
```

在这个例子中 Lookup 查找 Company 字段为 Sight Diver 的记录，并取出它的 Contact 和 Phone 两个字段的内容保存在 Variant 数组 v 中。程序里在显示结果之前先调用 VarIsArray 以确定 v 是一个数组，然后将此数组的两个元素取出并显示。如果只返回一个字段的内容，就会简单一些：

```
ShowMessage( String( ClientDataSet1.Lookup( 'Company', 'Sight Diver', 'Phone' ) ) );
```

这个例子只取出 Phone 字段的内容，所以简单得多了。

6.1.6 对记录排序

对于单向访问方式的数据集来说，只有一种记录排序方式，那就是在

CommandText 属性的 SQL 语句中加入 ORDER BY 子句，如：

```
SELECT COUNTRY, CURRENCY FROM COUNTRY ORDER BY CURRENCY
```

或：

```
SELECT COUNTRY, CURRENCY FROM COUNTRY ORDER BY COUNTRY DESC
```

所有的数据集控件都一样，在没有指定排序条件时，都是以主键为索引进行排序，这是因为大多数 RDBMS 都是这样实现 SQL 语句的。如果要对非主键字段排序，或要按逆序排序，一般有两种方法：

- 用 ORDER BY 子句；
- 用索引；

但是单向访问的数据集不支持索引排序，因为这要求对结果数据集的记录顺序作调整，要实现索引方式排序只能使用 Provider/Resolve 机制。

TClientDataSet 提供了两种索引排序的方法：

- 用 IndexFieldNames

  在 IndexFieldNames 属性中填入要排序的字段，可以有多个，以分号分隔，按先后顺序依次排序，即如果第一个字段相同时按第二个字段排，以此类推，这种方法的缺点是不能按逆序排。

- 用 IndexName

  在 IndexName 属性中填入要排序的索引，索引需要在 RDBMS 中用 CREATE INDEX 语句预先建立好才能用，在建立索引时可以自由选择多个字段及其相应的顺序（也可以逆序）。

注意：以上两种方法是互斥的，即二者只能选一种，指定一种后则自动清除另一种。

6.1.7 选择部分记录

一般来说应用程序往往只需要用到数据集的部分记录，这种情况下，就需要对数据集进行过滤，把符合特定条件的记录过滤出来。前面已经说了，对于单向访问的数据集，记录过滤比较不方便，要通过关闭数据集，修改 SQL 语句，再打开数据集来实现，但对于双向访问的数据集就要方便得多，只需要设置一下 Filter 和 Filtered 属性即可。

不过对于记录很多的数据集，而程序又只需要要用到很少的一部分，最好还是用修改 SQL 语句查询的方法，这样比较有效率，因为如果把所有记录数据都缓冲到 TClientDataSet 中再来过滤的话要花较多时间，并且也浪费资源。

对记录集进行过滤除了用前面所说的 Filter 属性外，还可以用 OnFilterRecord 事件响应。二者各有所长：

- Filter 属性使用简单方便，可以动态地改变过滤条件，但因为它是一个字符串，只支持简单的语法，不能进行复杂的过滤操作。虽然可以用运算符构成复合的条件表达式，但只限于几个常见的运算符。更主要的是，用 Filter 特性指定的表达式中只能出现数据集中已有的字段名和常量，而不能与程序中的变量结合；
- OnFilterRecord 事件相对麻烦一些，但可以在事件响应中用程序进行

复杂的过滤操作。它要比 Filter 属性灵活得多，能够在设计期就指定好过滤条件。在处理 OnFilterRecord 事件的响应中任意指定过滤条件，过滤条件可以很复杂；

Filter 属性的内容可以是固定的字符串，如前面的例子：

```
ClientDataSet1.Filter   := 'COUNTRY LIKE ''%a%'' ';
```

注意：其中使用了 LIKE 运算符，这是 RDBMS 才支持的，对于 Paradox 之类的数据库，BDE 提供的 Local SQL 不支持此语法，因为这里用的 dbExpress 只能连接 RDBMS，而且例子用的数据库是 InterBase，所以可以用它。

也可以将过滤条件由用户指定，如：

```
ClientDataSet1.Filter := Edit1.Text;
```

或：

```
ClientDataSet1.Filter := ‘COUNTRY = ‘ + Edit1.Text;
```

在上面一个例子中，用户可以自由输入过滤条件，而下面一个例子中，用户可以指定所要的国家名。

设置好过滤条件后，只要简单地将 Filter 属性设置为 True，即可将数据集中不符全条件的记录全部过滤掉。Filter 中可以用的运算符如表 3：

表 3：可以用于 Filter 中的逻辑运算符

<	小于	AND	逻辑与
>	大于	OR	逻辑或
>=	大于等于	NOT	逻辑非
<=	小于等于		
=	等于		
<>	不等于		

从上表可见 Filter 所支持的逻辑运算符与 DELPHI 本身差不多。

使用 OnFilterRecord 事件就不需要设置 Filter 属性了，但仍然要设置 Filtered 属性，如果设置了 OnFilterRecord 的事件响应，在 Filtered 属性设置为 True 时，每条记录都会触发 OnFilterRecord 事件。在 OnFilterRecord 事件响应函数中，有一个参数：Accept，在事件响应中将此参数设置为 True，表示此记录符合过滤条件，将保留；如果此参数为 False，则表示此记录不符合条件，将被过滤掉。

如下面这个例子：

```
Procedure TForm1.ClientDataSet1FilterRecord( DataSet : TDataSet; Var
Accept : Boolean );
Begin
```

```
    Accept := DataSet.['Company'] = 'Sight Diver';
End;
```

在这个例子中，所有 Company 字段不为 Sight Diver 的记录都会被过滤掉。

要浏览过滤后的记录集（它相当于原来的数据集的一个子集），可以用 TDataSet 的四个方法：

- FindFirst 定位当前记录到过滤后的数据集中的第一条记录；
- FindLast 定位当前记录到过滤后的数据集中的最后一条记录；
- FindNext 定位当前记录到过滤后的数据集中的下一条记录；
- FindPrior 定位当前记录到过滤后的数据集中的上一条记录；

上述方法在调用成功后都返回 True，否则返回 False。TDataSet 有一个 Found 属性可以确定上次调用是否成功。

如果设置了 Filter 属性或 OnFilterRecord 事件，即使 Filtered 属性为 False，调用上述四个方法也会暂时将 Filtered 属性设置为 True 然后移动当前记录，移动完成后恢复 Filtered 的状态。

如果没有设置 Filter 属性和 OnFilterRecord 事件，这四个方法与 First、Last、Next 和 Prior 是一样的。

6.1.8 MASTER/DETAIL 关系

对于一般数据库应用程序来说，Master/Detail（主/明细表）结构是一种非常典型的应用。主表中记录的一条信息对应了明细表中的多条记录，如有一个客户表，可能对应一个购物明细表，记录每个客户的购物情况；或者一个部门表对应一个员工表（部门明细表），记录每个部门的员工。对于这种结构，我们需要在程序中根据主表的记录变化及时更新明细表的显示，当然这可以用程序实现，响应主表的记录移动事件，在其中更新明细表的显示，但这是一个麻烦的办法，除非使用非数据感知控件显示，否则一般没必要这么用，Delphi 为 Master/Detail 结构提供了很方便的实现，可以不写一行代码。

从本质上说，Master/Detail 结构的实现是基于数据记录过滤技术的，明细表根据主表的变化修改过滤条件，将与主表相关的记录显示出来。对于 dbExpress 技术来说，有两种方法实现 Master/Detail：

- 在 TSQLDataSet 一级用 SQL 语句过滤；
- 在 TClientDataSet 一级用 MasterSource/MasterFields 过滤；

两种方法各有所长：

- 用 SQL 语句过滤比较麻烦，需要在主表变化时刷新 TClientDataSet，因为它缓冲了记录数据，不会自动随着 TSQLDataSet 的变化而变化，即使 TSQLDataSet 已关闭，TClientDataSet 仍然可以浏览数据，只是不能更新到数据库而已；但是这种方法只从数据库中取出必要的数据，当明细表的记录非常之多时，它的性能将比在 TClientDataSet 一级过滤要好很多。
- 用 TClientDataSet 的 MasterSource/MasterFields 过滤较为简单，只需要设置好相应的属性即可，一般不用写代码；但是因为这种方法在数据集打开时，要从数据库中取出明细表的所有记录缓冲在

TClientDataSet 中，再在这个缓冲数据集里进行过滤，所以在明细表的记录数很多时，性能较差。

下面以一个例子来说明用这两种方法如何来实现 Master/Detail 结构。首先是用 TClientDataSet 的 MasterSource/MasterFields 的例子，图 7 是这个例子的 Form：

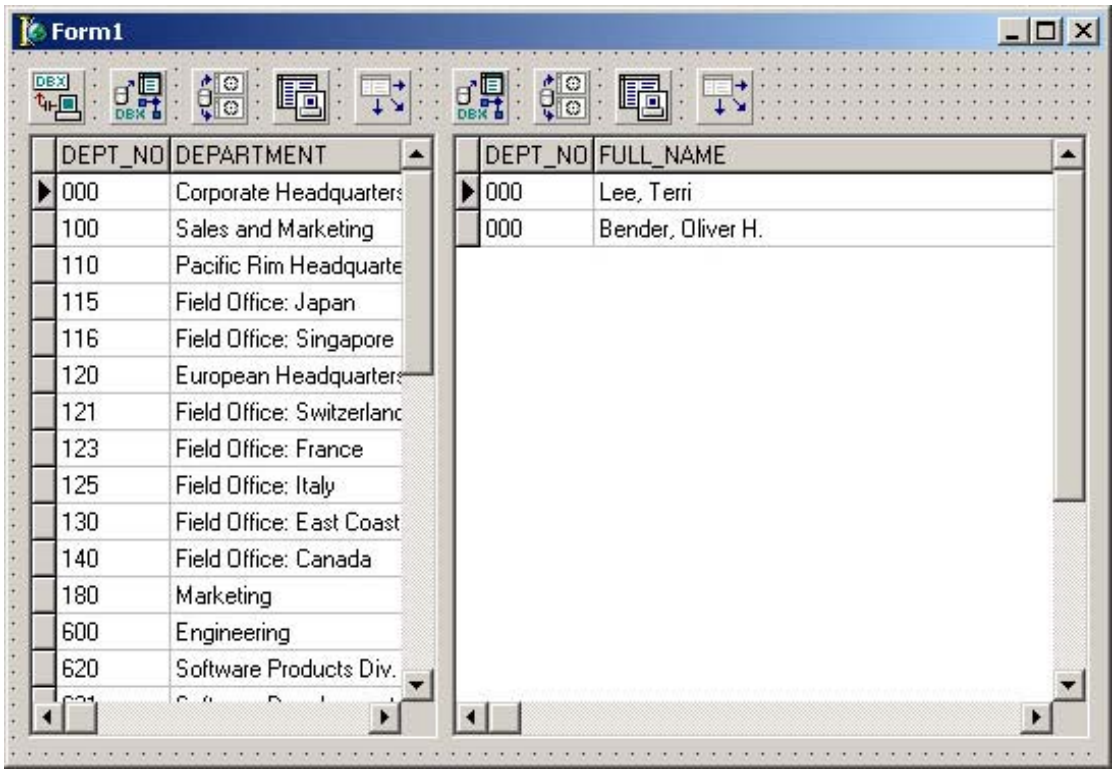

图 7：一个 Master/Detail 的例子

其中各控件的属性设置如表 4：

表 4：Master/Detail 的例子控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径>\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
	Connected	True（前提是先设置好上面的属性）
SQLDataSet1	CommandText	select DEPT_NO, DEPARTMENT from DEPARTMENT

	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet1	ProviderName	DataSetProvider1
	Active	True（前提是先设置好上面的属性）
DataSource1	DataSet	ClientDataSet1
DBGrid1	DataSource	DataSource1
SQLDataSet2	CommandText	select DEPT_NO, FULL_NAME from EMPLOYEE
	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
DataSetProvider2	DataSet	SQLDataSet2
ClientDataSet2	ProviderName	DataSetProvider2
	MasterSource	DataSource1
	MasterFields	DEPT_NO
	Active	True（前提是先设置好上面的属性）
DataSource2	DataSet	ClientDataSet2
DBGrid2	DataSource	DataSource2
其它控件	Caption	如图 7

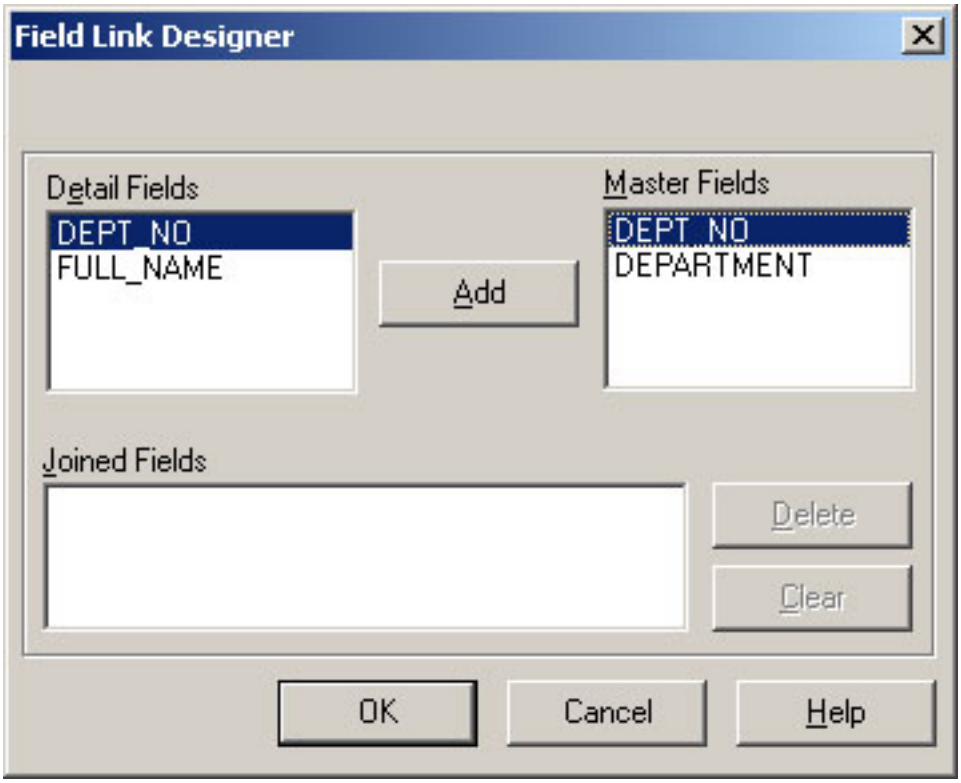

图 8：设置 MasterFields 关联

注意：其中 ClientDataSet2 的 MasterSource 为 DataSource1，通过这个属

性，将 ClientDataSet2 与 ClientDataSet1 联系起来。MasterFields 用于建立明细表与主表的关联字段，在 Object Inspector 中按 MasterFields 属性右边的小按钮即可打开字段关联设置对话框，如图 8 所示。在 Detail Fields 和 Master Fields 中分别选择明细表和主表中相关联的字段，按 Add 按钮即可将此关联加入 Joined Fields 列表中，如果主表和明细表之间有多个字段关联，可以依次将其加入，最后确定即可。

这样就完成了一个用 TClientDataSet 的 MasterSource/MasterFields 实现的 Master/Detail 结构应用的例子，不需要写一行代码，这个程序已经可以工作了。即使是在设计期间（程序未运行）卷动 DBGrid1，立即可以看到 DBGrid2 中记录的变化。

下面再来看一下用 TSQLDataSet 的 SQL 语句如何实现 Master/Detail 结构的应用。首先需要将表 4 的各控件属性改为如表 5 所示（其它控件属性不变）：

表 5：Master/Detail 的例子控件属性设置

控件	属性	属性值
SQLDataSet2	CommandText	select DEPT_NO, FULL_NAME from EMPLOYEE ***WHERE DEPT_NO = :DEPT_NO***
	SQLConnection	SQLConnection1
	DataSource	***DataSource1***
	Active	True（前提是先设置好上面的属性）
ClientDataSet2	ProviderName	DataSetProvider2
	MasterSource	***（为空即可）***
	MasterFields	***（为空即可）***
	Active	True（前提是先设置好上面的属性）

注意其中的变化之处（黑斜体的部分）：

- SQLDataSet2 的 CommandText 属性中的 SQL 语句增加了过滤条件(WHERE 子句)，并且有一个参数 DEPT_NO，这个参数就是与明细表关联的主表字段名，它会随着主表此字段值的变化而过滤出明细表中相应的内容。
- SQLDataSet2 的 DataSource 属性设置为 DataSource1，由此设置与明细表关联的主表，dbExpress 能够自动根据此主表的变化将主表当前记录的相应字段设置到 SQL 语句的参数中，这是用这种方法实现 Master/Detail 的一个关键。
- ClientDataSet2 的 MasterSource/MasterFields 清空即可，因为这时的过滤操作是由 SQLDataSet2 实现的。

但是这个例子与前面用 TClientDataSet 实现的不同，在设计期卷动 DBGrid1 不会导致 DBGrid2 的变化，这是因为 DBGrid1 的变化只是影响了 SQLDataSet2，但 ClientDataSet2 中缓冲了原来的记录数据，所以这里仍然显示的是原来的数据，不会随之变化。要让 DBGrid2 能随着 DBGrid1 的变化而更新，需要在 DBGrid1 变化后刷新 ClientDataSet2 中缓冲的记录数据。在 ClientDataSet1 的 AfterScroll 事件中添加如下代码：

```
procedure TForm1.ClientDataSet1AfterScroll(DataSet: TDataSet);
```

```
begin
    If ( ClientDataSet2.Active ) Then
        ClientDataSet2.Refresh;
end;
```

这段代码很简单，其功能就是在 ClientDataSet1 的当前记录变化（即在 DBGrid1 里改变当前记录）后更新 ClientDataSet2 的缓冲记录数据，因为 ClientDataSet1 先打开，为防止在 ClientDataSet2 未打开时刷新而出错，这里加上了刷新条件，必须在数据集打开时才执行此操作。

6.2 查询数据库

对于数据库应用程序来说，最为经常的操作应该就是数据查询—从大量的数据中找出所需要的那一部分。

6.2.1 有效地使用查询

在 dbExpress 中，最简单的查询就是通过 TSQLTable 控件打开一个数据表，并取出表中所有的数据记录。但这种方法不够灵活，局限性太大。

实际情况常常是要通过从多个表中联合查询才能取得所要的结果，用 TSQLTable 的方法就很难实现。虽然它也不是不能实现多表查询，通过 Lookup 字段等方法也是可以的，但局限性很大，而且不能进行复杂的查询。所以更通常的用法是用 TSQLDataSet 或 TSQLQuery 通过 SQL 语句实现。

6.2.2 可以查询哪些数据库

dbExpress 目前支持四种数据库：MySQL、InterBase、DB2、Oracle。因为 dbExpress 技术的要求所支持的数据库系统必须是跨平台的 RDBMS，所以它不支持任何文件型数据库或 Microsoft SQL Server 之类不支持跨平台的数据库。

由于 dbExpress 只支持 RDMBS，所以不像 BDE 那样，要考虑在访问文件型数据库时必须使用 Local-SQL（一种由 BDE 为文件型数据库实现的 SQL 语言的子集，只支持 SQL 的部分功能，详见 BDE 的 Local-SQL 帮助）。dbExpress 中用到的 SQL 语句都是由 RDMBS 处理的，只要 RDBMS 支持，可以在 dbExpress 中使用各种 SQL 语言的功能。

6.2.3 结构化查询语言-SQL

SQL 是结构化查询语言（Structured Query Language），通常读作 SEQUEL，是一种介于关系代数与关系演算之间的语言，集成了查询、操纵、定义和控制四个方面，是一个通用的，功能很强的关系数据库语言，目前是关系数据库的标准语言。

SQL 最早是由 IBM 的工程师提出，并且最早在 IBM 的实验数据库系统—System R 中实现，它的出现导致了关系型数据库管理系统（RDBMS）的出现，早期的代表就是 Oracle 和 DB2。1986 年美国国家标准局（ANSI）的数据库委员

会 X3H2 批准了以 SQL 作为数据库语言的美国标准，同年发布了 SQL 的标准文本。此后不久，国际标准化组织（ISO）也接受了 SQL 作为数据库语言的国际标准。目前最新的 SQL 标准是 ANSI SQL 92。

SQL 之所以能够为广大用户和厂商所接受，并成为国际标准，是因为它是一个综合的、通用的、功能很强而又简洁易学的语言。SQL 集数据查询（Data Query）、数据操纵（Data Manipulation）、数据定义（Data Definition）和数据控制（Data Control）等四大功能于一体，充分体现了 RDBMS 的优点。它的主要特点有：

- 综合统一：SQL 风格统一，可以独立完成数据库生命周期中的全部活动，包括定义关系模式、录入数据及建立、查询、更新、维护数据，数据重建、数据库安全控制等一系列操作要求，从而为数据库应用系统开发提供了良好的环境。
- 高度非过程化：用 SQL 进行数据操作时，用户只需要提出“做什么”，而不必明确说明“如何做”，因此用户无需了解数据的存取细节，用户只需要把 SQL 语句提供给 RDBMS 即可，系统会自动完成用户的要求，并返回结果。
- 面向集合的操作方式：SQL 采用集合操作方式，不仅查询的结果是数据记录集，而且可以以数据记录集为单位进行数据的插入、删除或更新。
- 以同一种语法结构提供两种使用方式：SQL 语言既是自含式语言，又是嵌入式语言。作为自含式语言，它可以在数据库系统提供的交互式 SQL 工具（如 InterBase 提供的 isql）中直接操作数据库。作为嵌入式语言，它可以通过高级语言（如 Delphi）向 RDBMS 发送请求，并取回执行结果（在 Delphi 中将这一操作通过控件实现）。
- 语言简洁易学：虽然 SQL 的功能很强大，但是它设计巧妙，语言十分简洁，其用于完成四种数据功能的核心只有 9 个动词：CREATE, DROP, ALTER, SELECT, INSERT, UPDATE, DELETE, GRANT, REVOKE。

SQL 由四个部分组成，即：数据查询语言（DQL），数据定义语言（DDL），数据操纵语言（DML），数据控制语言（DCL）。有时为了简单起见，也将 DQL 和 DML 合并称为 DML，将 DDL 和 DCL 合并称为 DDL 而分为两部分。

- 数据查询语言：只有一条 SQL 命令—SELECT，但是它的命令选项非常多，是所有 SQL 命令中最多的一条，可以说是一个命令集。它不但本身可以实现很复杂的功能，它还可嵌套或用于其它命令（如 DML）中。要用好此语句需要有一定的集合运算和关系代数知识。
- 数据定义语言：主要有三条 SQL 命令—CREATE、DROP 和 ALTER。分别用于创建、删除、修改元数据（即数据库结构对象，如表、视图、存储过程等）。
- 数据操纵语言：主要是 INSERT、UPDATE 和 DELETE 三条，有时也把 SELECT 算为 DML。主要用于完成对数据的操纵，如插入、更新、删除或查询。
- 数据控制语言：主要是用于数据库安全性控制的 GRANT 和 REVOKE，它们分别用来对数据库用户分配/回收权限。有时也将它们并入 DDL 中。

接下来主要对 SELECT 命令作一些简要介绍，所介绍的内容是大多数据 RDBMS 都支持的标准 SQL 语法，有不支持的地方将作一些说明。

用 SELECT 语句进行基本的数据库查询语法如下：

```
SELECT 字段 1, 字段 2, ...
FROM 表名
```

此 SQL 语句将把表的特定列的数据的全部记录列出来。如果要取出所有字段，也可以简单地用一个“*”号来代替所有的字段名。要过滤特定条件的记录，可以加上 WHERE 子句，如下：

```
SELECT *
FROM 表名
WHERE 条件
```

WHERE 子句中的条件可以是任何合法的 SQL 逻辑表达式，除了可以用前面表 3 所列的运算符以外，SQL 还可以用三个特别的运算符：

- LIKE：它可以用于字符串的模糊匹配，用法如：

```
SELECT *
FROM 表名
WHERE 字段 1 LIKE ‘x%’
```

这个语句表示查询所有字段 1 中以 x 开头的记录。条件中的%是用于匹配任何数量字符的通配符，类似 DOS 下用于匹配文件名的*。注意：使用%通配时必须使用 LIKE 运算符，如果用=则上例就表示查询完全匹配 x%的记录了。另外如果查询字符串中包括%，在 LIKE 的条件中应该用连续两个%来表示一个%字符。

- IN：用于用集合来匹配字段内容，例如：

```
SELECT *
FROM 表名
WHERE 字段 1 IN ( x， y， ... )
```

这个语句表示查询所有字段 1 在后面括号的集合中的记录。IN 也常用于嵌套查询，详见下面对嵌套查询的说明。

- BETWEEN..AND：用于查询介于二者之间的值，如：

```
SELECT *
FROM 表名
WHERE 字段 1 BETWEEN xx AND yy
```

SELECT 语句还可以使用 ORDER BY 子句来对查询结果进行排序，如下：

```
SELECT 字段 1, 字段 2
FROM 表名
ORDER BY 字段 2 [DESC]
```

ORDER BY 可以指定以某个列做排序，DESC 是可选的，表示逆序（从大到小）

排列，若没有指明或 ASC，则是从小到大排列。

SELECT 语句还可以实现多表查询。多表查询是指所查询得数据来源并不只有单一的表，而是联合一个以上的表才能够得到结果的，用法如：

```
SELECT *
FROM 表 1, 表 2
WHERE 表 1. 字段 1 = 表 2. 字段 1
```

多表查询时要注意两个表中关联字段的数据类型必须相同。上例中的联合条件是表 1 的字段 1 与表 2 的字段 1 相同，复杂的查询可能会用到多个表。多表查询还可以用 JOIN 进行表连接的方法，但是不同的 RDBMS 的 JOIN 语法略有不同，这里就不介绍了，详见相应的 RDBMS 说明文档。

SELECT 语句还可以在查询时进行统计，如：

```
SELECT COUNT (*)
FROM 表名
WHERE 字段名 = xxx
```

上例用于查询符合条件的数据记录共有几条。除了 COUNT 以外，还有可以对字段进行统计的 SUM（字段名）、AVG（字段名）、MAX（字段名）、MIN（字段名）等几个函数，分别用于计算总和、平均值、最大值、最小值。它们只能用于数字类型的字段。对于统计查询，还常常使用分组，如下：

```
SELECT 字段 1, AVG (字段 2 )
FROM 表名
GROUP BY 字段 1
HAVING AVG (字段 2 ) > xxx
```

GROUP BY 子句指定以列 1 为分组计算列 2 的平均值，HAVING 子句必须和 GROUP BY 一起使用作为统计的限制。

另外，SELECT 语句还可以嵌套（不过 MySQL 不支持此语法）。下面这个例子用于查询表 2 中存在符合条件的记录时，取出表 1 中的相应内容，通常表 2 的条件与表 1 有联系。

```
SELECT *
FROM 表 1
WHERE EXISTS (
SELECT *
FROM 表 2
WHERE 条件 )
```

另一种嵌套是集合型的，如下例是查找表 1 中字段 1 在表 2 的符合条件的字段 1 集合中的记录。在这种类型的应用中，表 2 只能 SELECT 一个字段，并且其类型要与表 1 中的相应字段相同，表 2 的过滤条件一般也是与表 1 有关联的：

```
SELECT *
FROM 表 1
WHERE 字段 1 IN (
SELECT 字段 1
FROM 表 2
WHERE 条件 )
```

以上所介绍的是 SELECT 命令的一些基本的用法，在绝大多数支持 SQL 标准的 RDBMS 中都能很好地运行，不过各种 RDBMS 都有一些各自的扩展，如 Oracle 能够以树形的方式进行查询等，另外，不同的 RDBMS 也有少许不相同之处，详见相应的 RDBMS 的手册说明。

6.2.4 参数化查询

作为最经常用到的查询操作，其本身的查询条件也是多变的，特别是其 WHERE 子句中的条件参数更是常常要根据程序的执行或用户的选择而改变，如果只是因为这么一点小小的变化就要重新生成 SQL 语句无疑是一件麻烦事。而且每次变更 SQL 语句，对于 RDBMS 来说就要在临时缓冲中重新编译这个语句，效率也会变低。

这种情况下就应该使用参数化查询。参数化查询就是把 SQL 语句中的不确定部分换成参数，在执行时才把它设置为具体的值。一个典型的参数化查询的例子如：

```
SELECT * FROM Table1 WHERE ID = :ID
```

其中的:ID 就是参数，SQL 的语法中把所有以:（冒号）开头的标识符都认为是参数。当 RDBMS 收到这样的带参数的 SQL 语句将会把它编译到临时缓冲中，在执行前把参数的值赋给参数，然后执行即可，变更条件时只要传不同的参数即可，比重新生成 SQL 语句要简单得多，而且因为 SQL 语句未变，所以 RDBMS 不用再重新编译 SQL 语句，效率也会高一些。

在 TSQLDataSet 中如何使用参数化查询呢？首先，要将 CommandText 属性中的 SQL 语句设为带参数的 SQL 语句；然后再在 Params 属性中设置参数的值；最后打开数据集即可。

注意：如果 TSQLDataSet 的 ParamCheck 属性为 False 时将不会自动根据 SQL 语句的变化调整 Params 属性，如果不手工加入相应参数将导致参数找不到的异常，所以建议将 ParamCheck 属性设为 True。

下面用一个例子来说明在 dbExpress 中如何使用参数化查询。图 9 是这个例子的 Form：

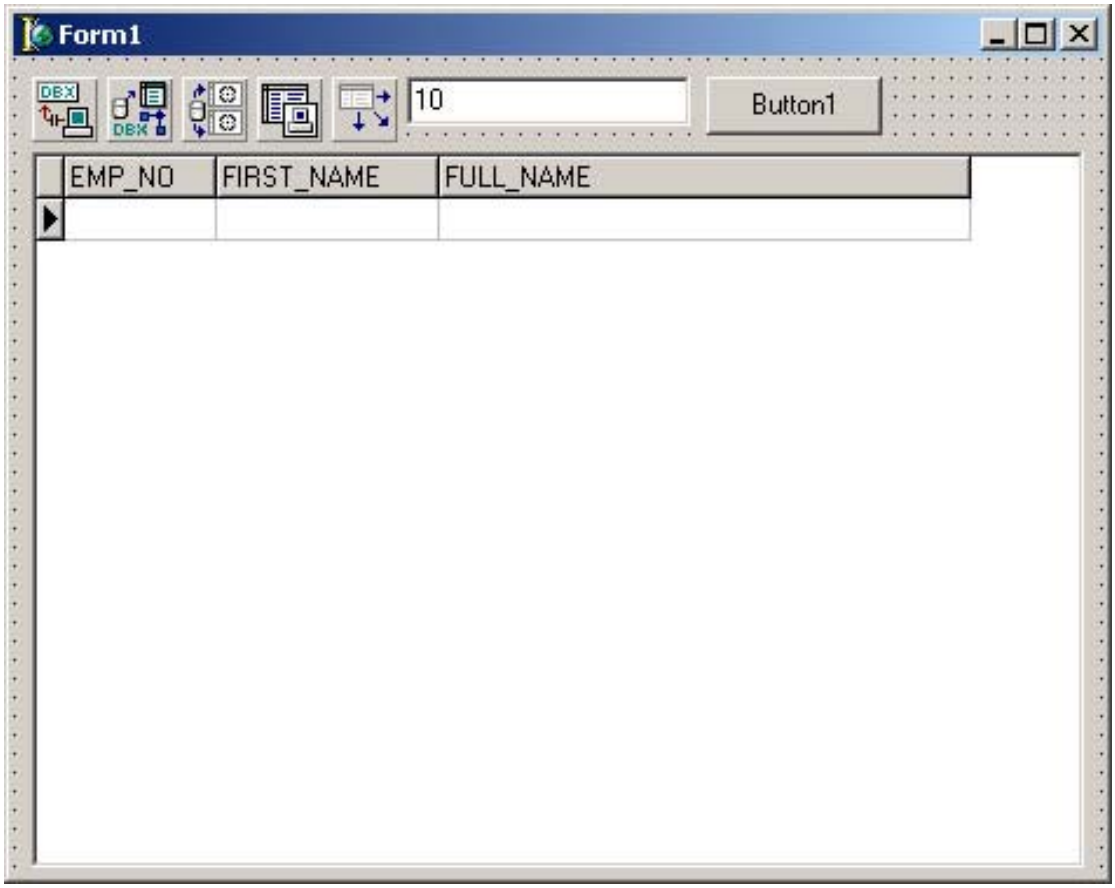

图 9：一个参数化查询的例子

其中各控件的属性设置如表 6：

表 6：参数化查询的例子控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径>\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
	Connected	True（前提是先设置好上面的属性）
SQLDataSet1	CommandText	select EMP_NO, FIRST_NAME, FULL_NAME from EMPLOYEE WHERE EMP_NO < :EMP_NO
	SQLConnection	SQLConnection1

	Active	True（前提是先设置好上面的属性）
DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet1	ProviderName	DataSetProvider1
	Active	True（前提是先设置好上面的属性）
DataSource1	DataSet	ClientDataSet1
DBGrid1	DataSource	DataSource1
Edit1	Text	‘10’
Button1		
其它控件	Caption	如图 9

最后按后面的程序清单设置事件响应，并写入程序代码。完整的程序清单如下：

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, DBXpress, FMTBcd, StdCtrls, Grids, DBGrids, DB, DBClient,
  Provider, SqlExpr;

type
  TForm1 = class(TForm)
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
    DataSetProvider1: TDataSetProvider;
    ClientDataSet1: TClientDataSet;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Edit1: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
    SQLDataSet1.Close;
    SQLDataSet1.Params.ParamByName(    'EMP_NO'    ).AsInteger    :=
StrToInt( Edit1.Text );
    SQLDataSet1.Open;
    ClientDataSet1.Refresh;
end;

end.
```

这段程序的关键是在 Button1 的事件响应中关闭 SQLDataSet1，然后根据 Edit1 中输入的数据设置参数值，然后重新打开 SQLDataSet1 并刷新 ClientDataSet1 的缓冲内容即可看到改变参数后的查询结果。

参数化查询不仅仅用于查询语句，也常常用于其它 DML 语句，包括 INSERT、DELETE、UPDATE。如用下面的语句可以实现向 COUNTRY 表中插入一条记录，其内容为用户在 Edit1 中输入的数据：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    SQLDataSet1.Close;
    SQLDataSet1.CommandText    :=    'INSERT    INTO    COUNTRY    VALUES
( ''PRC'', :CURRENCY ) ';
    SQLDataSet1.Params.ParamByName(    'CURRENCY'    ).AsString    :=
Edit1.Text;
    SQLDataSet1.ExecSQL;
end;
```

注意：上例中用了 ExecSQL 方法，因为 INSERT 语句是一个不返回结果的语句，只需要执行它，而没有数据集可返回，所以这里不是用 Open 方法，而是用 ExecSQL 方法。对于 DELETE 和 UDPATE 这种没有返回结果的语句也是应该用 ExecSQL 方法，而不能用 Open 方法。

6.2.5 执行查询

从前面的介绍中可以知道，执行一个查询语句就是调用数据集控件的 Open 方法或者用与之等效的将 Active 属性设为 True 来打开数据集实现的。在数据集打开时，会向 RDBMS 发出 CommandText 属性中的 SELECT 语句，RDBMS 执行完 SELECT 语句后就会把结果记录集送回数据集控件，然后由数据集控件将数据通过 DataSource 控件等提供给数据感知控件显示。

对于非查询 SQL 语句的执行则略有不同，因为这种 SQL 语句没有返回结果，

执行此种语句只需要将 SQL 语句提交给 RDBMS 执行即可，所以不用打开数据集，只要调用 ExecSQL 方法，将 SQL 语句送给 RDBMS 执行。

下例说明二者的不同：

```
begin
    SQLDataSet1.Close;
    SQLDataSet1.CommandText := ‘UPDATE COUNTRY SET CURRENCY = ‘’RMB’’ WHERE
COUNTRY = ‘’PRC’’ ’;  //  假设已经将 COUNTRY=’PRC’的记录加入表中
    SQLDataSet1.ExecSQL;
    SQLDataSet1.CommandText := ‘SELECT COUNTRY, CURRENCY FROM COUNTRY ‘;
    SQLDataSet1.Open;
end;
```

6.2.6 查询结果

对于 INSERT 之类的 SQL 语句是没有查询结果的，但是对于一个 SELECT 语句，如果存在符合条件的记录，则会返回一个记录集合，存在于数据集控件中，对于使用 Provider/Resolve 机制的 dbExpress 应用程序来说，使用的是缓冲在 TClientDataSet 中的记录集。下面来说明一下如何使用缓冲在 TClientDataSet 中的数据集。

顾名思义，所谓数据集就是由若干条数据记录所组成的集合。其中的记录数可以大于 0 也可以为 0，如果查询结果为空，即不存在符合查询条件的记录，则此数据集中的记录数为 0，也就是说是一个空集。

在 Delphi 中，一次只能操作一条记录（这也是大多数编程语言操作数据集的方法），这条可以被访问的记录称为当前记录。要访问其它的记录，必须通过数据集浏览方法：First、Last、Next、Prior 等移动当前记录。一条记录通常会有很多字段，Delphi 将每个字段的信息记录在一个 TField 对象中，TField 是一个基类，实际上 Delphi 会自动根据每个字段的数据类型产生相应类型的 TField 对象，如对于整数类型的字段，会产生 TIntegerField，对于字符串类型的字段产生 TStringField，不过它们都是 TField 类型的派生类，这里统一称为 TField 类型。

TField 类型提供了一组属性、方法和事件用于对字段数据进行操作。最常用的属性是一组类型转换属性，以 As 开头，如 AsInteger（将字段值转为整数类型返回）、AsString（将字段值转为字符串类型返回），所用的转换类型不一定就是字段的实际类型，如可以对整数类型的字段使用 AsString，它将自动把字段的值转为一个字符串，相当于调用了 IntToStr 函数，但是这种转换不一定都能成功，如对一个内容不为数字的字符串类型字段使用 AsInteger 将导致一个类型转换失败的异常。

下面用一个例子来说明 TField 类型的使用。这个例子没有使用数据感知控件，而是用程序从 TField 中取出数据并显示到一个 TListView 控件中。图 10 为此例的 Form：

图 10：一个用 TField 取出查询结果数据集的例子

其中各控件的属性设置如表 7：

表 7：用 TField 取出查询结果数据集的例子控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径>\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
	Connected	True（前提是先设置好上面的属性）

SQLDataSet1	CommandText	SELECT E.EMP_NO, E.FULL_NAME, D.DEPARTMENT FROM EMPLOYEE E, DEPARTMENT D WHERE D.DEPT_NO = E.DEPT_NO ORDER BY D.DEPT_NO, E.EMP_NO
	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
ListView1	ViewStyle	vsReport
	Columns	按 Object Inspector 中此属性右边的小按钮或在 ListView1 上右击鼠标，在弹出的菜单中选择【Columns Editor ...】。在打开的对话框中增加三个列，分别将其 Caption 设置为‘员工号’、‘员工姓名’和‘所在部门’，另外要将它们的宽度设置为合适的值。

最后按后面的程序清单设置事件响应，并写入程序代码。完整的程序清单如下：

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, ComCtrls, DBXpress, DB, SqlExpr, FMTBcd;

type
  TForm1 = class(TForm)
    ListView1: TListView;
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
Var
    p : TListItem;
begin
    With SQLDataSet1 Do
    Begin
        Open;
        While ( Not Eof ) Do
        Begin
            p := ListView1.Items.Add;
            p.Caption := FieldByName( 'EMP_NO' ).AsString;
            p.SubItems.Add( FieldByName( 'FULL_NAME' ).AsString );
            p.SubItems.Add( FieldByName( 'DEPARTMENT' ).AsString );
            Next;
        End;
        Close;
    End;
end;

end.
```

这个程序用了一个比前面的例子略为复杂的 SQL 语句：

```
SELECT E.EMP_NO, E.FULL_NAME, D.DEPARTMENT
FROM EMPLOYEE E, DEPARTMENT D
WHERE D.DEPT_NO = E.DEPT_NO
ORDER BY D.DEPT_NO, E.EMP_NO
```

在这个 SQL 语句中用了两个表的联合查询，并且由于两个表存在同名字段，所以必须给每个字段加上表名前缀，为简单起见给每个表用了一个字母的别名。另外这个查询还对结果进行了排序，先按部门排序，再对部门内按员工号排序。

因为这里不用数据感知控件，也不需要支持双向数据访问，只需要能把数据集全部取出即可，所以没有用 Provider/Resolve 机制，而是直接操作 SQLDataSet 控件。程序在打开 SQLDataSet1 后就依次取出第一条记录的所有字段加入 ListView1 中，然后循环下移当前记录并将其各字段加入，直到数据集的末尾，最后关闭数据集。

至此，此表中的所有数据记录已经显示在 ListView1 中了，如图 11 所示：

图 11：用 TField 取出查询结果数据集的例子的运行结果

6.3 存储过程(StoredProcedure)

存储过程是一组在 RDBMS 服务端运行的 SQL 语句，是一种用 SQL 语言写的程序块。将多个 SQL 语句写成一个存储过程，可以很方便地调用执行，而且一般 RDBMS 都可以预先对存储过程进行编译，执行效率也会较好，使用存储过程是进行 C/S 结构数据库应用开发的重要手段。不过存储过程也有一些缺点，过多地使用存储过程会加重数据库服务器的负担，特别是在存储过程中进行复杂运算时，另外，不同的 RDBMS 的存储过程实现差别较大，基本上是不兼容的，这也就严重影响了应用系统的移植性。

本书以 InterBase 的为例对存储过程作一简要介绍，下面是一个典型的 InterBase 存储过程：

```
CREATE PROCEDURE ADD_COUNTRY (
  COUNTRY VARCHAR(15),
  CURRENCY VARCHAR(10)
)  AS
 DECLARE VARIABLE n INTEGER;
BEGIN
 SELECT COUNT(*) FROM COUNTRY WHERE COUNTRY = :COUNTRY INTO :n;
```

```
 IF ( n > 0 ) THEN
  UPDATE COUNTRY SET CURRENCY = :CURRENCY WHERE COUNTRY = :COUNTRY;
 ELSE
  INSERT INTO COUNTRY VALUES( :COUNTRY, :CURRENCY );
END
```

这个存储过程的用途是更新 COUNTRY 表中的指定 COUNTRY 记录的 CURRENCY 字段，如果此记录不存在则增加此 COUNTRY 记录。这个存储过程有两个字符串型的参数，分别为国家名（COUNTRY）和货币名（CURRENCY），定义了一个内部整型变量 n，共有三个 SQL 语句。首先执行一个 SELECT 语句检查表中是否有此 COUNTRY 的记录，如果有，则执行 UPDATE 语句更新其 CURRENCY 字段，否则执行 INSERT 语句插入此记录。

至于如何在 InterBase 中创建存储过程，详见后面的说明。

除了这种可执行存储过程外，还有一种返回数据集的存储过程，称为可查询的存储过程，关于这种存储过程见后面关于返回数据集的存储过程部分。

6.3.1 使用 TSQLStoredProc 控件的一般步骤

在程序中执行存储过程是通过 TSQLStoredProc 控件实现的，这个控件与一般的数据集控件一样，是从 TDataSet 派生出来的。它的最主要的三个属性是：SQLConnection、StoredProcName 和 Params，分别用于设置与数据相连的 SQLConnection 控件、存储过程名及其需要的参数。

因为 TSQLStoredProc 通常只用于执行不返回数据集的存储过程，所以用法比其它数据集控件要简单得多，只需要在执行前设置必要的输入参数，在执行后取出所需的执行结果，一般是以输出参数的形式返回。

另外，也可以用 TSQLDataSet 执行存储过程，只要将 CommandType 设置为 ctStoredProc，然后在 CommandText 中设置存储过程名即可像 TSQLStoredProc 一样使用。

6.3.2 在 InterBase 中写 StoredProcedure

要在 InterBase 数据库中创建存储过程，可以在 Delphi 所配的 SQL Explorer 工具中操作：建立一个 InterBase 别名，并将其连接到所需的数据库上，在 SQL 页中执行相应的语句即可。

注意：不能在 InterBase 6 所配的 IB Console 里的交互式 SQL 工具中直接使用存储过程创建语句，因为在此工具中，分号是默认的语句终止符，而存储过程的创建语句中通常会含有多个分号，所以执行语句时会出错。要在交互式 SQL 工具中建立存储过程必须按下面的做法：

```
SET AUTODDL OFF;
SET TERM ^ ;

CREATE PROCEDURE ADD_COUNTRY (
  COUNTRY VARCHAR(15),
```

```
  CURRENCY VARCHAR(10)
)  AS
 DECLARE VARIABLE n INTEGER;
BEGIN
 SELECT COUNT(*) FROM COUNTRY WHERE COUNTRY = :COUNTRY INTO :n;
 IF ( n > 0 ) THEN
  UPDATE COUNTRY SET CURRENCY = :CURRENCY WHERE COUNTRY = :COUNTRY;
 ELSE
  INSERT INTO COUNTRY VALUES( :COUNTRY, :CURRENCY );
END^

SET TERM ; ^
COMMIT WORK;
SET AUTODDL ON;
```

这个语句比前面的语句多了一些内容，SET AUTODDL 是用于设置是否自动提交数据定义语句，SET TERM 是用于设置语句终止符。因为分号要用于存储过程中，所以要暂时将交互式 SQL 工具的语句终止符改为其它字符，这里是改成‘^’字符，因为这个修改是暂时的，创建完存储过程就要改回来，所以不能让存储过程的创建自动提交，要到把语句终止符改回来以后再提交此次操作，所以还要暂时关闭 DDL 自动提交功能。

关于 DDL 自动提交和语句终止符设置，也可以在 Windows 版的交互式 SQL 工具中的选项里修改。

InterBase 的存储过程建立的基本语法如下：

```
CREATE PROCEDURE 存储过程名
    [(输入参数表)]
    [RETURNS (返回值表)]
AS
    [DECLARE VARIABLE 变量 类型;]
    ...
BEGIN
  语句...
END
```

关于建立存储过程的更详细的语法请参见本书的附录部分及 InterBase 帮助文档。

6.3.3 StoredProcedure 的参数

虽然存储过程不是必须要有参数的，但作为在服务端运行的程序都是一些功能性的实现，其操作大多数需要一些参数的，一般来说存储过程中的 SQL 程序也都是一些参数化查询语句，所以带参数的存储过程是很常见的。下面主要介绍如何操作一个带参数的存储过程。

存储过程的参数主要有两种：输入参数和输出参数，一个存储过程可以同时有这两种参数。

前面那个例子是一个典型的带输入参数的存储过程例子，下面来看一个带输出参数的例子：

```
CREATE PROCEDURE GET_EMP_COUNT
(
  DEPT_NO INTEGER
)
RETURNS
(
  EMP_COUNT INTEGER
)
AS
BEGIN
 SELECT COUNT(*) FROM EMPLOYEE WHERE DEPT_NO = :DEPT_NO INTO :EMP_COUNT;
 EXIT;
END
```

这个存储过程用于从员工表中统计各部门的员工数，它含有一个输入参数：部门号，和一个输出参数：员工数。

6.3.4 执行 StoredProcedure

在交互式 SQL 工具中执行存储过程是用 EXECUTE PROCEDURE 命令，如下：

```
EXECUTE PROCEDURE ADD_COUNTRY ‘PRC’, ‘RMB’
```

即可在表 COUNTRY 中加入一条 COUNTRY 为 PRC 的记录（如果没有此记录的话）或将 COUNTRY 为 PRC 的记录的 CURRENCY 字段改为 RMB（如果有此记录的话）。

对于可查询的存储过程则是像表一样使用，通过 SELECT 语句实现，如：

```
SELECT * FROM GET_ALL_EMP( 120 );
```

其中 GET_ALL_EMP 就是一个返回数据集的可查询存储过程。

在程序中使用可查询存储过程的方法与表的查询方法基本相同，只要在 CommandText 中设置类似于上面的 SQL 语句即可，详见后面关于返回数据集的存储过程部分。

6.3.5 返回数据集的 StoredProcedure

有的时候会需要存储过程返回一个数据集，这种可查询的存储过程相对于可执行的存储过程差别比较大，特别是调用这种存储过程的方法很像对表的操

作。而且由于它返回的是一个数据集，与一般的带输出参数的存储过程也很不一样。

下面是一个简单的返回数据集的存储过程例子：

```
CREATE PROCEDURE GET_ALL_EMP
(
  DEPT_NO INTEGER
)
RETURNS
(
  FULL_NAME VARCHAR(50)
)
AS
BEGIN
 FOR
  SELECT FULL_NAME FROM EMPLOYEE WHERE DEPT_NO = :DEPT_NO INTO :FULL_NAME
 DO
  SUSPEND;
END
```

这个存储过程乍一看很像带输出参数的存储过程，但它与带输出参数的存储过程有一个很大的区别是使用了 FOR 语句。对于一个用过其它 RDBMS 的人来说，看到这个存储过程一定会比较惊讶，因为在这个存储过程里用了一个 FOR 循环来逐条返回记录，而大多数其它 RDBMS（如 Microsoft SQL Server）是可以直接返回一个记录集的，这是因为 InterBase 的操作单位与它们不同，InterBase 全部都是以记录为单位进行操作的，对于后面将要讨论的触发器来说，也是如此。

下面将以一个完整的例子来说明前所说的关于存储过程的有关内容：图 12 为此例的 Form：

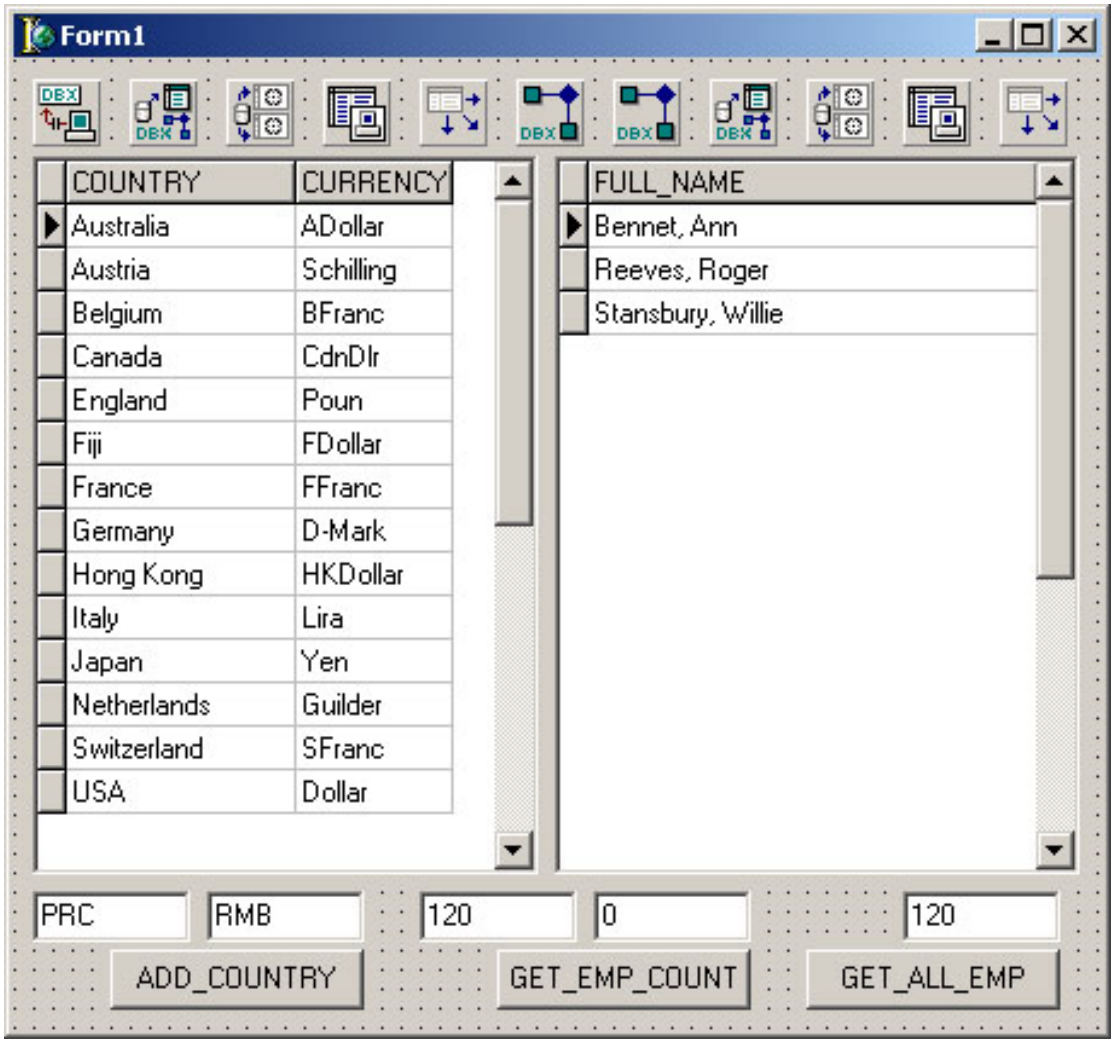

图 12：一个使用存储过程的例子

其中各控件的属性设置如表 8：

表 8：使用存储过程的例子控件属性设置

控件	属性	属性值
SQLConnection1	ConnectionName	IBLocal
	LoginPrompt	False
	Params	Database 设置为<InterBase 的安装路径>\Examples\Database\employee.gdb；这是 InterBase 提供的一个例子库，请根据自己的 InterBase 安装路径设置上面的路径； 如果修改过 SYSDBA 的密码，请相应修改 Password
	Connected	True（前提是先设置好上面的属性）

SQLDataSet1	CommandText	select COUNTRY, CURRENCY from COUNTRY
	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet1	ProviderName	DataSetProvider1
	Active	True（前提是先设置好上面的属性）
DataSource1	DataSet	ClientDataSet1
DBGrid1	DataSource	DataSource1
SQLStoredProc1	SQLConnection	SQLConnection1
	StoredProcName	ADD_COUNTRY
SQLStoredProc2	SQLConnection	SQLConnection1
	StoredProcName	GET_EMP_COUNT
SQLDataSet2	CommandText	SELECT * FROM GET_ALL_EMP(:DEPT_NO) 注意，这里要给 DEPT_NO 参数一个默认值，否则数据集打不开
	SQLConnection	SQLConnection1
	Active	True（前提是先设置好上面的属性）
DataSetProvider2	DataSet	SQLDataSet2
ClientDataSet2	ProviderName	DataSetProvider2
	Active	True（前提是先设置好上面的属性）
DataSource2	DataSet	ClientDataSet2
DBGrid2	DataSource	DataSource2
其它控件	Caption	如图 12

最后按后面的程序清单设置事件响应，并写入程序代码。完整的程序清单如下：

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, DBXpress, FMTBcd, Grids, DBGrids, DB, DBClient, Provider,
  SqlExpr, StdCtrls;

type
  TForm1 = class(TForm)
    SQLConnection1: TSQLConnection;
    SQLDataSet1: TSQLDataSet;
    SQLStoredProc1: TSQLStoredProc;
```

```
    DataSetProvider1: TDataSetProvider;
    ClientDataSet1: TClientDataSet;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    Edit3: TEdit;
    Button2: TButton;
    SQLStoredProc2: TSQLStoredProc;
    Edit4: TEdit;
    DBGrid2: TDBGrid;
    SQLDataSet2: TSQLDataSet;
    DataSetProvider2: TDataSetProvider;
    ClientDataSet2: TClientDataSet;
    DataSource2: TDataSource;
    Edit5: TEdit;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
    SQLStoredProc1.ParamByName( 'COUNTRY' ).AsString  := Edit1.Text;
    SQLStoredProc1.ParamByName( 'CURRENCY' ).AsString := Edit2.Text;
    SQLStoredProc1.ExecProc;
    ClientDataSet1.Refresh;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
    SQLStoredProc2.ParamByName(      'DEPT_NO'      ).AsInteger      :=
StrToInt( Edit3.Text );
    SQLStoredProc2.ExecProc;
    Edit4.Text := SQLStoredProc2.ParamByName( 'EMP_COUNT' ).AsString;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    SQLDataSet2.ParamByName(       'DEPT_NO'       ).AsInteger       :=
StrToInt( Edit5.Text );
    ClientDataSet2.Refresh;
end;

end.
```

这个程序演示了关于存储过程的三个方面内容：

- 带输入参数的存储过程：ADD_COUNTRY 按钮的事件响应执行时将设置存储过程 ADD_COUNTRY 的两个参数，然后执行此存储过程，最后刷新 ClientDataSet1，即可以 DBGrid1 中看到新增加的在 Edit1 和 Edit2 中输入的内容。或者如果此记录已存在则更新之。
- 带输出参数的存储过程：GET_EMP_COUNT 按钮的事件响应执行时将设置存储过程 GET_EMP_COUNT 的部门号参数，然后执行它，最后从返回的输出参数中取得结果，并在 Edit4 中显示。
- 返回数据集的存储过程：GET_ALL_EMP 按钮的事件响应执行时只是简单地设置了 SQLDataSet2 的部门号参数，然后刷新 ClientDataSet2。它是通过在 SQLDataSet2 中调用一个可查询的存储过程 GET_ALL_EMP 实现的。

6.3.6 特殊的 StoredProcedure – 触发器(Trigger)

触发器与存储过程很相似，不过比较特殊一些，它不是被别的程序所调用或执行，它是自动执行的，而且它是与具体的表相关的。可以这么说，触发器是在某个表进行某些操作时自动执行的一种特殊的存储过程。

InterBase 的触发器可以设置为在表进行 INSERT、UPDATE、DELETE 时触发，也可以设置在操作之前还是之后触发，还可以同时设置多个触发器在同一操作上并且设置其触发顺序。

要注意的是 InterBase 的触发器是基于记录触发的，即在记录集操作时，每一条记录的操作都会引起触发器触发，这一点与其它数据库较为不同，多数数据库都是以操作为单位触发的，即每次操作触发一次，而不管是否是记录集操作。InterBase 的这一特点虽然有时候会比较灵活，但因为对每条记录触发，所以处理不当，很容易降低系统性能。

在 InterBase 中触发器最典型的应用就是用于产生唯一主键，因为 InterBase 不像 Microsoft SQL Server 那样支持 identity，不过用生成器（Generator）配合触发器可以实现 identity 不能实现的功能，比如多个表使用同一生成器可以实现多个表的联合主键唯一。

下面是一个用于主键生成的触发器。

```
CREATE TRIGGER SET_EMP_NO FOR EMPLOYEE BEFORE INSERT POSITION 0 AS
BEGIN
    new.emp_no = gen_id(emp_no_gen, 1);
END
```

其中 SET_EMP_NO 为触发器名，EMPLOYEE 为触发此触发器的表名，BEFORE INSERT 决定了触发此触发器的操作，POSITION 决定此触发器在所有响应此表此操作的触发器中的顺序。

new 表示即将插入的记录，与之相应的还有一个 old，可以用于在删除操作触发时表示即将被删除的记录。emp_no_gen 为一个生成器，1 是生成器的增量，生成器相当于数据库中的一种全局变量，每次调用 gen_id 函数据操作生成器会使生成器按增量增加。创建生成器的方法如下：

```
CREATE GENERATOR 生成器名;
```

在 EMPLOYEE 表中建立了上述触发器后，执行下面的语句：

```
INSERT INTO EMPLOYEE ( FIRST_NAME, LAST_NAME, SALARY, DEPT_NO, JOB_CODE,
JOB_GRADE, JOB_COUNTRY )
VALUES( 'Hello', 'Raptor', 80000, 600, 'Dir', 2, 'USA' )
```

注意其中并没有 EMP_NO 字段的内容，但是执行下面的查询语句：

```
SELECT * FROM EMPLOYEE WHERE FIRST_NAME = ‘Hello’
```

可以找到这条记录，并且其 EMP_NO 被自动赋予了一个值，如 149，如果再插入一条记录，则此值将自动被赋为 150。

还有另一个被自动赋值的字段是 HIRE_DATE，它不是因为触发器而是因为它有一个默认值 Now，即如果插入的记录不含此字段内容，则将自动赋值为当前日期时间，这与触发器不同。

关于触发器的更详细语法说明见本书附录部分。