32장 DDG 버그 보고용 도구 - 데스크탑 애플리케이션 개발

* 이번 애플리케이션의 전반적인 요구 사항
* 데이터 모델
* 데이터 모듈 작성
* 사용자 인터페이스 작성
* 웹과의 연동 준비 작업

이번 장에서는 데스크탑 데이터베이스 애플리케이션의 개발 기법에 대해 이야기하겠다. 이번 장에서 이야기하는 DDG 버그 보고용 애플리케이션을 통해서 인터넷과의 연동을 염두에 둔 애플리케이션 설계 기법을 배울 수 있을 것이다. 또한, 데이터 조작 기능과 사용자 인터페이스를 분리시킬 때 발생하는 몇 가지 까다로운 문제들을 해결하는 방법들도 배울 수 있을 것이다.

델파이로 데이터베이스 애플리케이션을 개발하는 것은 그리 어렵지 않다. 그러나 너무 성급하게 애플리케이션을 설계/개발하면 나중에 그 애플리케이션을 좀더 개선시키려고 할 때 난관에 부딪힐 수가 있다는 점을 염두에 두어야 한다. 이번 장에서는 그러한 점을 염두에 두고 데이터베이스 애플리케이션을 만드는 방법을 제시하겠다.

이번 애플리케이션의 전반적인 요구 사항

그럼 DDG 버그 보고용 애플리케이션의 전반적인 요구 사항들을 짚어 보자. 단, 이번 예제에서 초점을 두는 것은 실질적인 버그 보고용 도구의 구현이 아니란 점을 이해하기 바란다. 이번 장에서 중점적으로 이야기하고자 하는 것은 앞에서 이야기했던 여러 가지 기법들이다. 따라서, 일반적인 버그 보고용 도구에 꼭 필요한 요구 사항이라도 이번 장의 주제에 벗어나는 것들은 과감히 생략했다.

웹에 대한 준비 - 웹 애플리케이션으로의 전환을 염두에 두어야 한다.

이번 버그 보고용 애플리케이션은 최소한의 노력만으로도 웹 애플리케이션으로 개선시킬 수 있어야 한다. 그렇게 하기 위해 가장 필요한 것은 데이터베이스 작업과 사용자 인터페이스를 "완전히" 분리시키는 것이다. 33장, "DDG 버그 보고용 도구 - 웹 브로커 활용"에서는 이번 장의 애플리케이션을 웹 페이지를 통해서 사용할 수 있도록 만들 것이다.

사용자 데이터 입력/로그온

버그 보고용 애플리케이션은 개별적인 하나의 테이블(Users.db)을 이용해서 시스템에 로그온할 수 있는 사용자들을 판별한다. 그 테이블에 속한 사용자들은 이 애플리케이션을 통해서 버그를 보고할 수 있으며, 또 다른 사용자를 이 애플리케이션에 추가할 수도 있다. 이번 버전의 애플리케이션에서는 사용자 정보를 수정하거나 삭제하는 기능은 생략했다.

사용자가 로그온할 때에는 사용자 이름(username)을 제공해야 하는데, 유효한 사용자 이름들은 Users.db 테이블에 저장되어 있다. 사용자 로그온은 보고된 버그들을 조작할 때 필요한 UserID를 얻기 위한 것일 뿐 보안을 고려한 것은 아니다.

버그 조작, 조회, 필터링

사용자는 버그 정보를 추가, 수정, 삭제할 수 있다. 하나의 버그 정보는 여러 가지 필드들로 구성된다. 예를 들어서, 버그를 보고하는 날짜라든지 보고 대상(즉, 해당 버그의 처리를 담당할 개발진), 버그 처리 상태, 버그에 대한 요약, 자세한 버그 내용, 관련 소스 등의 정보가 포함된다. 사용자 로그온 과정이 있기 때문에 버그를 입력한 보고자 정보는 자동적으로 추가된다.

버그에 대한 주석

기존의 버그에 주석(버그에 대한 추가 정보. 이 애플리케이션에서는 Action이라고 부른다)을 추가할 수 있다. 사용자는 자신 또는 다른 사용자가 추가한 주석들을 조회할 수 있다. 주석을 추가할 수 있으면 버그 처리 과정의 관리가 편해지며, 버그에 관련된 사람들이 버그에 대한 의견을 서로 주고 받는 데에도 사용할 수 있다.

기타 U/I 기능들

애플리케이션은 사용자가 이해하거나 사용하기 쉬운 사용자 인터페이스를 가져야 한다. 참조 필드(lookup field) 같은 기능들을 제공해야 하며, 레이블들도 최대한 이해하기 쉽게 작성해야 한다.

데이터 모델

그림 32.1은 이번 버그 보고용 애플리케이션의 데이터 모델이다. 데이터 모델을 구성하는 각 테이블들에 대한 설명은 다음과 같다.

- IDs : IDs 테이블은 키 생성 테이블의 역할을 하며, Users, Bugs, Actions 테이블에 사용할 키를 제공한다.
- Users : Users 테이블에는 버그 보고 시스템에 추가된 사용자들이 저장된다.
- Bugs : Bugs 테이블에는 버그에 대한 정보가 저장된다.
- Actions : Actions 테이블에는 버그에 대한 주석(추가 정보)이 저장된다. 하나의 버그가 여러 개의 주석을 가질 수 있다.
- Status : Status 테이블은 각 버그의 상태에 배정된 참조(lookup) 테이블이다.

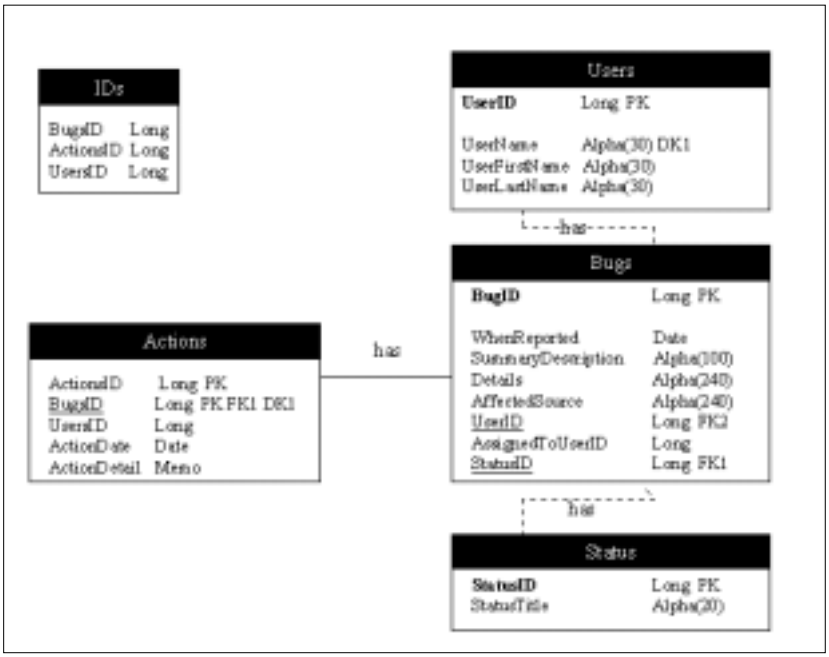


그림 32.1
버그 보고용 응용 프로그램의 데이터 모델

데이터 모듈 작성

데이터 모듈은 이번 버그 보고용 애플리케이션의 핵심부이다. 모든 데이터베이스 조작 작업은 데이터 모듈을 통해서 일어난다. 사용자 인터페이스는 데이터 모듈의 메소드와 속성들을 통해서 데이터 모듈의 기능을 사용한다. 사용자 인터페이스 요소가 데이터 조작 컴포넌트를 직접 참조하는 경우는 없다. 단, 사용자 인터페이스 폼에 존재하는 TDataSource 컴포넌트의 DataSet 속성처럼 오브젝트 인스펙터로 직접 설정하는 경우는 예외이다.



데이터 조작 기능과 사용자 인터페이스를 분리시킬 때 TDataSource 컴포넌트를 어디에 두느냐를 가지고 논쟁하기 좋아하는 개발자들이 있는데, 그런 것은 중요하지 않다고 생각한다. 이번 장에서는 TDataSource를 데이터 모듈이 아니라 U/I 폼에 포함시켰는데, TDataSource가 데이터 조작보다는 사용자 인터페이스 쪽에 더 관련이 많다고 생각했기 때문이다. 그러나 이것은 단순히 취향일 뿐이다. TDataSource를 데이터 모듈에 놓는다고 해도 달라질 것은 없다.

목록 32.1은 버그 보고용 애플리케이션의 데이터 모듈에 대한 소스 코드이다.


목록 32.1 Ch32\DDGBugsDM.pas

```
unit DDGBugsDM;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Db, DBTables, HTTPApp, DBWeb;

type

  EUnableToObtainID = class(Exception);
```

```
TDDGBugsDataModule = class(TDataModule)
  dbDDGBugs: TDatabase;
  tblBugs: TTable;
  tblUsers: TTable;
  tblStatus: TTable;
  tblActions: TTable;
  tblBugsBugID: TIntegerField;
  tblBugsWhenReported: TDateField;
  tblBugsSummaryDescription: TStringField;
  tblBugsDetails: TStringField;
  tblBugsAffectedSource: TStringField;
  tblBugsUserID: TIntegerField;
  tblBugsStatusID: TIntegerField;
  dsUsers: TDataSource;
  dsStatus: TDataSource;
  tblIDs: TTable;
  tblBugsUserNameLookup: TStringField;
  tblBugsAssignedToLookup: TStringField;
  tblUsersUserID: TIntegerField;
  tblUsersUserName: TStringField;
  tblUsersUserFirstName: TStringField;
  tblUsersUserLastName: TStringField;
  tblBugsAssignedToUserID: TIntegerField;
  dsBugs: TDataSource;
  wbdpBugs: TWebDispatcher;
  dstpBugs: TDataSetTableProducer;
  procedure DDGBugsDataModuleCreate(Sender: TObject);
  procedure tblBugsBeforePost(DataSet: TDataSet);
  procedure tblBugsFilterRecord(DataSet: TDataSet; var Accept: Boolean);
  procedure tblUsersBeforePost(DataSet: TDataSet);
  procedure tblBugsAfterInsert(DataSet: TDataSet);
  procedure wbdpBugswaBrowseAllBugsAction(Sender: TObject;
    Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  procedure wbdpBugswaIntroAction(Sender: TObject; Request: TWebRequest;
    Response: TWebResponse; var Handled: Boolean);
  procedure wbdpBugswaUserNameAction(Sender: TObject;
```

```
    Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  procedure wbdpBugswaVerifyUserNameAction(Sender: TObject;
    Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
private
  FLoginUserID: Integer;
  FLoginUserName: String;

  function GetFilterOnUser: Boolean;
  procedure SetFilterOnUser(const Value: Boolean);
  function GetNumBugs: Integer;
protected
  procedure PostAction(Sender: TObject; Action: TStrings);
public

  // Bugs 테이블 관련 메소드들
  procedure FirstBug;
  procedure LastBug;
  procedure NextBug;
  procedure PreviousBug;
  function IsLastBug: Boolean;
  function IsFirstBug: Boolean;
  function IsBugsTblEmpty: Boolean;
  procedure InsertBug;
  procedure DeleteBug;
  procedure EditBug;
  procedure SaveBug;
  procedure CancelBug;
  procedure SearchForBug;

  // Users 테이블 관련 메소드들
 {$IFNDEF DDGWEBBUGS}
  procedure AddUser;
 {$ENDIF}

  procedure PostUser(Sender: TObject);
  function GetUserFLName(AUserID: Integer): String;
```

```
    // Actions 테이블 관련 메소드들

    {$IFNDEF DDGWEBBUGS}
    procedure AddAction;
    {$ENDIF}

    procedure GetActions(AActions: TStrings);

    // ID 생성 메소드들
    function GetDataSetID(const AFieldName: String): Integer;
    function GetNewBugID: Integer;
    function GetNewUserID: Integer;
    function GetNewActionID: Integer;

    // 로그인 함수
    function Login: Boolean;

    // 노출시킬 속성들
    property LoginUserID: Integer read FLoginUserID;
    property FilterOnUser: Boolean read GetFilterOnUser write
          SetFilterOnUser;
    property NumBugs: Integer read GetNumBugs;
  end;

var
  DDGBugsDataModule: TDDGBugsDataModule;

implementation

{$IFNDEF DDGWEBBUGS}
uses UserFrm, ActionFrm;
{$ENDIF}

{$R *.DFM}

// 범용 루틴들
function IsInteger(IntVal: string): Boolean;
var
```

```
  v, code: Integer;
begin
  val(IntVal, v, code);
  Result := code = 0;
end;

procedure MemoFromStrings(AMemoField: TMemoField; AStrings: TStrings);
var
  Stream: TMemoryStream;
begin
  Stream := TMemoryStream.Create;
  try
    AStrings.SaveToStream(Stream);
    Stream.Seek(0, soFromBeginning);
    AMemoField.LoadFromStream(Stream);
  finally
    Stream.Free;
  end;
end;

procedure StringsFromMemo(AStrings: TStrings; AMemoField: TMemoField);
var
  Stream: TMemoryStream;
begin
  Stream := TMemoryStream.Create;
  try
    AMemoField.SaveToStream(Stream);
    Stream.Seek(0, soFromBeginning);
    AStrings.LoadFromStream(Stream);
  finally
    Stream.Free;
  end;
end;

// 내부 메소드들
function TDDGBugsDataModule.GetFilterOnUser: Boolean;
begin
  Result := tblBugs.Filtered;
```

```
end;

procedure TDDGBugsDataModule.SetFilterOnUser(const Value: Boolean);
begin
 tblBugs.Filtered := Value;
end;

function TDDGBugsDataModule.GetNumBugs: Integer;
begin
  Result := tblBugs.RecordCount;
end;

// ID 생성 메소드들
function TDDGBugsDataModule.GetDataSetID(const AFieldName: String):
Integer;
const
  MaxAttempts = 50;
var
  Attempts: Integer;
  NextID: Integer;
begin
  tblIDs.Active := True;
  // 50번을 시도해도 성공하지 못한 채로 루프가 끝나면 루프 다음의
 // if문에서 예외를 발생시킨다.
  Attempts := 0;
  while Attempts <= MaxAttempts do
  begin
    try
      Inc(Attempts);
      // 다른 사용자가 테이블을 수정하고 있으면 여기서 에러가 발생한다.
      tblIDs.Edit;
      // Break()가 수행된다는 것은 Edit() 메소드가 성공했다는 뜻이다.
      // Break()가 수행되면 실행 흐름은 루프를 벗어난다.
      Break;
    except
      on EDBEngineError do
      begin
        // 약간의 시간 지연
```

```
        Continue;
      end;
    end;
  end;

  if tblIDs.State = dsEdit then
  begin
     // 테이블에서 ID를 얻는다. 그리고 테이블의 값을 1 증가시킨다. 그 값은
     // 다음 레코드가 사용하게 된다.
    NextID := tblIDs.FieldByName(AFieldName).AsInteger;
    tblIDs.FieldByName(AFieldName).AsInteger := NextID + 1;
    TblIDs.Post;
    Result := NextID;
  end
  else
    Raise EUnableToObtainID.Create('Cannot create unique ID');
end;

function TDDGBugsDataModule.GetNewActionID: Integer;
begin
  Result := GetDataSetID('ActionsID');
end;

function TDDGBugsDataModule.GetNewBugID: Integer;
begin
  Result := GetDataSetID('BugsID');
end;

function TDDGBugsDataModule.GetNewUserID: Integer;
begin
  Result := GetDataSetID('UsersID');
end;

// 초기화/로그인 메소드들

procedure TDDGBugsDataModule.DDGBugsDataModuleCreate(Sender: TObject);
begin
 { 주/종 관계가 실패하지 않게 하기 위해서는 다음과 같은 순서로 테이블들을
```

```
    열어야 한다. }

  ssnBugs.Active := True;
  dbDDGBugs.Connected := True;
  tblUsers.Active  := True;
  tblStatus.Active := True;
  tblBugs.Active   := True;
  tblActions.Active := True;

end;


function TDDGBugsDataModule.Login: Boolean;
var
  UserName: String;
begin
  InputQuery('Login', 'Enter User Name: ', UserName);
  Result := tblUsers.Locate('UserName', UserName, []);
  if Result then
  begin
    FLoginUserID   := tblUsers.FieldByName('UserID').AsInteger;
    FLoginUserName := tblUsers.FieldByName('UserName').AsString;
  end;
end;

// Bugs 테이블 관련 메소드들

procedure TDDGBugsDataModule.FirstBug;
begin
  tblBugs.First;
end;

procedure TDDGBugsDataModule.LastBug;
begin
  tblBugs.Last;
end;

procedure TDDGBugsDataModule.NextBug;
```

```
begin
  tblBugs.Next;
end;

procedure TDDGBugsDataModule.PreviousBug;
begin
  tblBugs.Prior;
end;

function TDDGBugsDataModule.IsLastBug: Boolean;
begin
  Result := tblBugs.Eof;
end;

function TDDGBugsDataModule.IsFirstBug: Boolean;
begin
  Result := tblBugs.Bof;
end;

function TDDGBugsDataModule.IsBugsTblEmpty: Boolean;
begin
  // RecordCount가 0이면 테이블에 저장된 버그 정보가 하나도 없는 것이다.
  Result := tblBugs.RecordCount = 0;
end;

procedure TDDGBugsDataModule.InsertBug;
begin
  tblBugs.Insert;
end;

procedure TDDGBugsDataModule.DeleteBug;
var
  Qry: TQuery;
  BugID: Integer;
begin
  if MessageDlg('Delete Action?', mtConfirmation, [mbYes, mbNo], 0) =
     mrYes then
  begin
```

```
    BugID := tblBugs.FieldByName('BugID').AsInteger;
      // 동적으로 생성한 TQuery 컴포넌트를 이용해서 버그 레코드를 삭제한다.
    Qry := TQuery.Create(self);
    try
      dbDDGBugs.StartTransaction;
      try
        // 우선, 이 버그에 속한 주석들을 모두 삭제한다.
        Qry.DatabaseName := dbDDGBugs.DataBaseName;
        Qry.SQL.Add(Format('DELETE FROM ACTIONS WHERE BugID = %d', [BugID]));
        Qry.ExecSQL;

        // 그런 다음, Bugs 테이블에서 버그를 삭제한다.
        Qry.SQL.Clear;
        Qry.SQL.Add(Format('DELETE FROM BUGS WHERE BugID = %d', [BugID]));
        Qry.ExecSQL;

        tblBugs.Refresh;
        tblActions.Refresh;

        dbDDGBugs.Commit;
      except
        dbDDGBugs.Rollback;
        raise;
      end;
    finally
      Qry.Free;
    end;
  end;
end;

procedure TDDGBugsDataModule.EditBug;
begin
  tblBugs.Edit;
end;

procedure TDDGBugsDataModule.SaveBug;
begin
  tblBugs.Post;
```

```
end;

procedure TDDGBugsDataModule.CancelBug;
begin
  tblBugs.Cancel;
end;

procedure TDDGBugsDataModule.SearchForBug;
var
  BugStr: String;
begin
  InputQuery('Search for bug', 'Enter bug ID: ', BugStr);
  if IsInteger(BugStr) then
    if not tblBugs.Locate('BugID', StrToInt(BugStr), []) then
      MessageDlg('Bug not found.', mtInformation, [mbOK], 0);
end;

// Users 테이블 관련 메소드들

{$IFNDEF DDGWEBBUGS}
procedure TDDGBugsDataModule.AddUser;
begin
  tblUsers.Insert;
  try
    if NewUserForm(PostUser) = mrCancel then
      tblUsers.Cancel;
  except
    // 에러가 발생했으므로 테이블을 조회(browse) 모드로 전환하고
   // 예외를 발생시킨다.
    tblUsers.Cancel;
    raise;
  end;
end;
{$ENDIF}

procedure TDDGBugsDataModule.PostUser(Sender: TObject);
begin
  if tblUsers.State = dsInsert then
```

```
    tblUsers.FieldByName('UserID').AsInteger := GetNewUserID;
  tblUsers.Post;
end;

function TDDGBugsDataModule.GetUserFLName(AUserID: Integer): String;
begin
  // 이름과 성을 결합해서 돌려준다.
  if tblUsers.Locate('UserID', AUserID, []) then
    Result := Format('%s %s',
[tblUsers.FieldByName('UserFirstName').AsString,
      tblUsers.FieldByName('UserLastName').AsString])
  else
    Result := EmptyStr;
end;

{$IFNDEF DDGWEBBUGS}
procedure TDDGBugsDataModule.AddAction;
begin
  NewActionForm(PostAction);
end;
{$ENDIF}

procedure TDDGBugsDataModule.GetActions(AActions: TStrings);
var
  Action: TStringList;
  ActionUserId: Integer;

begin
  Action := TStringList.Create;
  try
    with tblActions do
    begin
      tblActions.First;
      while not Eof do
      begin
        Action.Clear;
        ActionUserID := FieldByName('UserID').AsInteger;
        StringsFromMemo(Action, TMemoField(FieldByName('ActionDetail')));
```

```
        AActions.Add(Format('Action Added on: %s',
                   [FormatDateTime('mmm dd, yyyy',
          FieldByName('ActionDate').AsDateTime)]));
        AActions.Add(Format('Action Added by: %s',
                   [GetUserFLName(ActionUserID)]));
        AActions.Add(cPrefix+EmptyStr);
        AActions.AddStrings(Action);
        AActions.Add(cPrefix+'==============================');
        AActions.Add(cPrefix+EmptyStr);
        tblActions.Next;
      end; // while
    end; // with
  finally
    Action.Free;
  end;
end;

procedure TDDGBugsDataModule.PostAction(Sender: TObject; Action:
 TStrings);
var
  BugID: Integer;
begin
  tblActions.Insert;
  try
    BugID := tblBugs.FieldByName('BugID').AsInteger;
    tblActions.FieldByName('ActionID').AsInteger      := GetNewActionID;
    tblActions.FieldByName('BugID').AsInteger         := BugID;
    tblActions.FieldByName('UserID').AsInteger        := LoginUserID;
    tblActions.FieldByName('ActionDate').AsDateTime   := Date;
    MemoFromStrings(TMemoField(tblActions.FieldByName('ActionDetail')),
          Action);
    tblActions.Post;
  except
    tblActions.Cancel;
    raise;
  end;
end;
```

```
// 이벤트 처리 메소드들
procedure TDDGBugsDataModule.tblBugsBeforePost(DataSet: TDataSet);
begin
  if tblBugs.State = dsInsert then
    tblBugs.FieldByName('BugID').AsInteger := GetNewBugID;
end;

procedure TDDGBugsDataModule.tblBugsFilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
begin
  Accept := tblBugs.FieldByName('UserID').AsInteger = FLoginUserID;
end;

procedure TDDGBugsDataModule.tblUsersBeforePost(DataSet: TDataSet);
begin
  if tblUsers.State = dsInsert then
    tblUsers.FieldByName('UserID').AsInteger := GetNewUserID;
end;

procedure TDDGBugsDataModule.tblBugsAfterInsert(DataSet: TDataSet);
begin
  tblBugs.FieldByName('UserID').AsInteger := FLoginUserID;
  tblBugs.FieldByName('UserNameLookup').AsString := FLoginUserName;
end;

 end.
```

메인 폼이 나타나기 전에 사용자 로그인 대화상자를 띄우기 위해서 프로젝트 파일을 목록 32.2와 같이 수정했다.

목록 32.2 Ch32\DDGBugs.pas

```
program DDGBugs;
uses
  Forms,
  Dialogs,
  ChildFrm in 'ChildFrm.pas' {ChildForm},
```

```
  DBModeFrm in 'DbModeFrm.pas' {DBModeForm},
  DBNavStatFrm in 'DBNavStatFrm.pas' {DBNavStatForm},
  MainFrm in 'MainFrm.pas' {MainForm},
  UserFrm in 'UserFrm.pas' {UserForm},
  ActionFrm in 'ActionFrm.pas' {ActionForm},
  DDGBugsDM in 'DDGBugsDM.pas' {DDGBugsDataModule: TDataModule};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TDDGBugsDataModule, DDGBugsDataModule);
  if DDGBugsDataModule.Login then
  begin
    Application.CreateForm(TMainForm, MainForm);
    Application.Run;
  end
  else
    MessageDlg('Invalid Login', mtError, [mbOk], 0);
end.
```

애플리케이션의 초기화와 로그인

목록 32.2를 보면 가장 먼저 TDDGBugsDataModule을 생성한 다음, TDDGBugsDataModule의 Login() 메소드를 호출한다. Login()의 수행 결과에 따라서 애플리케이션의 실행을 계속하거나 중지시킨다. 사용자가 로그인 대화상자에 입력한 사용자 이름이 Users.db에 존재하면 if문이 참이 되므로 메인 폼이 뜨지만, 사용자 이름이 존재하지 않으면 메시지 상자를 표시한 후 애플리케이션의 실행을 종료한다.

ID 생성

버그 보고용 애플리케이션은 파라독스(Paradox) 데이터베이스를 백 엔드로 사용한다. 버그 정보들을 제대로 관리하려면 각 테이블들의 각 레코드들에는 키값으로 사용할 고유한 ID를 부여해야 하는데, 파라독스의 자동 증가(auto-increment) 필드를 키 필드

로 사용하면 쉽게 해결할 수 있을 것 같지만, 필자의 데이터베이스 개발 경험에 비추어 볼 때 파라독스 자동 증가 필드의 신뢰성은 매우 낮다. 특히, 외부키들에 대해서는 절대 사용하지 않는 것이 좋다. 그래서 이번 애플리케이션은 IDs.db 테이블을 이용해서 직접 고유한 키값을 생성해낸다.

IDs.db 테이블에는 Bugs, Users, Actions 테이블의 키로 사용할 정수값이 저장된다. TDDGBugsDataModule.GetDataSetID() 메소드는 IDs 테이블의 값을 한 번 읽은 후 그 값을 1 증가시키기 때문에 IDs.db 테이블의 값은 항상 유일한 값이 된다. 그리고 GetDataSetID()는 IDs.db 테이블을 사용하는 사용자가 하나도 없을 때에만 ID값을 조작하기 때문에 두 사용자에게 동일한 키값을 돌려주는 경우도 없다. Bugs, Users, Actions 테이블은 각각 GetNewBugID(), GetNewUserID(), GetNewActionID()를 통해서 자신의 ID를 얻는데, 그 세 메소드는 GetDataSetID()의 AFieldName 인자에 자신의 필드 이름을 넣고 GetDataSetID()를 호출한다. 각 테이블에 어떤 레코드를 보내려면(post) 그 전에 ID를 얻어야 하므로 tblBugs와 tblUsers의 BeforePost 이벤트 처리 메소드에서 각각 GetNewBugID()와 GetNewUserID()를 호출해서 ID를 얻고 레코드의 적절한 필드에 그 ID값을 설정한다. Actions 테이블의 경우에는 PostAction() 메소드에서 GetNewActionID()를 호출한다.

Bugs 테이블 조작 메소드들

"// Bugs 테이블 관련 메소드들"이라는 주석 아래에 나오는 메소드들이 Bugs 테이블을 다루는 메소드들이다. 레코드 이동 메소드들은 특별히 설명할 것이 없다. DeleteBug()가 그나마 가장 복잡한 메소드인데, 이 메소드는 버그 레코드 자체를 삭제하기 전에, 우선 그 버그 레코드에 관련된 주석(action) 레코드들을 모두 삭제한다. 버그 레코드와 주석 레코드들의 삭제는 TDataBase의 트랜잭션 기능을 이용해서 단일한 트랜잭션으로 처리된다. 따라서, 하나라도 에러가 있으면 전체 삭제 작업이 취소되고, 삭제된 모든 레코드들은 원상태로 복구된다. 파라독스 같은 지역 데이터베이스에 대해 트랜잭션 처리를 수행하는 경우에는 이번 예제에서처럼 TDatabase의 TransIsolation 속성을 tiDirtyRead로 해야 한다는 점을 기억하기 바란다.

버그 레코드들의 조회/필터링

사용자는 모든 버그들을 볼 수도 있고, 자신이 보고한 버그들만 볼 수도 있다. 자신이 보고한 버그들만을 골라내는 기능은 tblBugs 컴포넌트의 Filtered 속성을 사용해서 구현한다. tblBugs.Filtered 속성을 True로 설정하면 각각의 레코드에 대해 tblBugs.

OnFilterRecord 이벤트가 발생한다. OnFilterRecord 이벤트 처리 메소드에서 현재 로그인한 사용자의 ID를 뜻하는 전역 저장 필드 FLoginUserID와 각 레코드의 UserID 필드를 비교하면 그 사용자가 보고한 버그들만을 골라낼 수 있다. 그런데 tblBugs 컴포넌트의 Filtered 속성은 폼(즉, 사용자 인터페이스 부분)이 직접 조작하지 않는다. 이번 장의 주요 목적 중의 하나가 모든 데이터 조작 기능을 하나의 데이터 모듈에 집중화시키는 방법을 보여주는 것이므로 tblBugs.Filtered 속성도 TDDGBugsDataModule. FilterOnUser 속성을 통해서 조작하도록 했다. 물론, 폼에서 직접 데이터 모듈 안에 있는 테이블 컴포넌트의 속성이나 이벤트에 접근해도 문법적으로 틀린 것은 아니다. 애플리케이션의 구조상의 문제일 뿐이다.

사용자 추가

사용자 추가에 관련된 메소드는 TDDGBugsDataModule.AddUser()와 TDDGBugsDataModule.PostUser() 메소드이다. AddUser() 메소드는 사용자를 추가하기 위한 간단한 대화상자를 띄운다. 이때 사용자 입력 폼을 띄우는 역할을 하는 NewUserForm()이라는 메소드의 인자에 PostUser() 메소드를 넣는다. NewUserForm()은 인자로 넘겨 받은 메소드를 이용해서 테이블에 사용자를 추가하는데, 이런 식으로 메소드를 넘겨주는 이유는 데이터 모듈이 띄운 폼에서 데이터 모듈의 메소드를 직접 호출할 수 있게 하기 위한 것이다. 물론, 그러한 기능을 구현하는 데 사용할 수 있는 기법은 지금 설명한 것 이외에도 여러 가지가 있겠지만, 그냥 메소드 주소를 넘겨주는 방식이 무난하다는 생각이 든다. NewUserForm()이 띄우는 사용자 추가 폼은 목록 32.3에 나온 UserFrm.pas에 정의되어 있다.

목록 32.3 Ch32\UserFrm.pas

```
unit UserFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask, DBCtrls;

type

  TUserForm = class(TForm)
```

```
    lblUserName: TLabel;
    dbeUserName: TDBEdit;
    lblFirstName: TLabel;
    dbeFirstName: TDBEdit;
    lblLastName: TLabel;
    dbeLastName: TDBEdit;
    btnOK: TButton;
    btnCancel: TButton;
    procedure btnOKClick(Sender: TObject);
  private
    FPostUser: TNotifyEvent;
  public
    { Public 선언부 }
  end;

function NewUserForm(APostUser: TNotifyEvent): Word;

implementation
uses dbTables;
{$R *.DFM}

function NewUserForm(APostUser: TNotifyEvent): Word;
var
  UserForm: TUserForm;
begin
  UserForm := TUserForm.Create(Application);
  try
    UserForm.FPostUser := APostUser;
    Result := UserForm.ShowModal;
  finally
    UserForm.Free;
  end;
end;

procedure TUserForm.btnOKClick(Sender: TObject);
begin
  if dbeUserName.Text = EmptyStr then begin
    MessageDlg('A user name is required.', mtWarning, [mbOK], 0);
    dbeUserName.SetFocus;
```

```
      ModalResult := mrNone;
    end
    else begin
      try
        FPostUser(self);
      except
        on EDBEngineError do
        begin
          MessageDlg('User name already exists.', mtWarning, [mbOK], 0);
          dbeUserName.SetFocus;
          ModalResult := mrNone;
        end;
      end;
    end;
  end;

  end.
```

NewUserForm()은 TUserForm의 인스턴스를 생성해서 화면에 띄운다. APostUser 인자는 TNotifyEvent형의 FPostUser 저장 필드에 넣는다. FPostUser는 TNotifyEvent형의 메소드 포인터이며, TDDGBugsDataModule의 PostUser() 역시 TNotityEvent형과 동일한 인자 구성을 가지고 있으므로 FPostUser에 TDDGBugsDataModule.PostUser() 메소드의 주소를 직접 넣을 수 있다. 이러한 기법은 컴포넌트 작성에 대해 설명하는 20장, "비주얼 컴포넌트 라이브러리의 핵심 요소들"과 21장, "델파이 사용자 컴포넌트 만들기"에서 설명한 바 있다.

사용자가 OK 버튼을 클릭하면 btnOKClick() 메소드가 호출된다. 사용자 정보는 DBEdit 컴포넌트들(Users.db 테이블의 필드들과 연결되어 있다)에 입력되어 있기 때문에 FPostUser(PostUser()를 가리키고 있음)를 호출하면 입력된 사용자 정보가 Users 테이블에 저장된다. 만일 입력된 사용자 이름이 이미 존재하는 것이면 예외가 발생한다. 그런데 PostUser() 메소드는 TUserForm 안에서 호출하기 때문에, 결과적으로 데이터 모듈 안에서 발생한 예외를 TUserForm 안에서 처리할 수 있게 된다는 점을 주목하기 바란다. 이처럼 예외를 중첩시키는 것은 컴포넌트를 설계할 때 흔히 적용하는 기법이다. 이런 식으로 예외 처리 구조를 만들면 데이터 모듈을 다른 요소로부터 완전히 독립적으로 캡슐화할 수 있다.

주석의 추가

주석(소스에서는 Action이라고 지칭한다)은 각각의 버그에 추가되는 참고 정보이다. 사용자는 다른 사용자들이 보고한 버그에도 자신의 주석을 추가할 수 있다. TDDGBugsDataModule.AddAction() 메소드는 주석 추가 폼을 띄우는 NewActionForm() 이라는 메소드를 호출한다. 목록 32.4는 주석 추가 폼과 NewActionForm() 메소드를 정의하는 ActionFrm.pas의 소스 코드이다.

목록 32.4 Ch32\ActionFrm.pas

```
unit ActionFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type

  TPostActionEvent = procedure (Sender: TObject; Action: TStrings) of
      Object;

  TActionForm = class(TForm)
    memAction: TMemo;
    lblAction: TLabel;
    btnOK: TButton;
    btnCancel: TButton;
    procedure btnOKClick(Sender: TObject);
  private
    FPostAction: TPostActionEvent;
  public
    { Public 선언부 }
  end;


procedure NewActionForm(APostAction: TPostActionEvent);

implementation
```

```
{$R *.DFM}

procedure NewActionForm(APostAction: TPostActionEvent);
var
  ActionForm: TActionForm;
begin
  ActionForm := TActionForm.Create(Application);
  try
    ActionForm.FPostAction := APostAction;
    ActionForm.ShowModal;
  finally
    ActionForm.Free;
  end;
end;

procedure TActionForm.btnOKClick(Sender: TObject);
begin
  if Assigned(FPostAction) then
    FPostAction(Self, memAction.Lines);
end;

 end.
```

NewActionForm() 메소드의 작동 방식은 NewUserForm()과 본질적으로 동일하다. 단, NewUserForm()이 인자로 받아들이는 메소드형은 델파이의 TNotifyEvent가 아니라 이번 예제에서 정의한 TPostActionEvent라는 것인데, 이 메소드형은 TObject형의 인자뿐만 아니라 주석 텍스트를 뜻하는 TStrings형의 인자도 받아들인다. 사용자가 OK 버튼을 클릭하면 btnOKClick() 이벤트 처리 메소드가 호출된다. btnOKClick() 메소드는 TDDGBugsDataSource.PostAction()을 호출해서 주석을 데이터베이스에 추가한다(FPost Action은 PostActions()를 가리키고 있다).

이것으로 데이터 모듈과 그에 관련된 폼들의 작성이 끝났다. 다음 장에서는 이 데이터 모듈에 코드를 추가해서 웹 버전의 버그 보고용 애플리케이션(ISAPI 서버)에 맞게 개선할 것이다.

사용자 인터페이스 작성

그럼 이번 버그 보고용 애플리케이션의 사용자 인터페이스를 작성하는 단계로 넘어가자. 웹 버전으로의 업그레이드를 염두에 두고 사용자 인터페이스를 작성하겠다.

메인 폼

사용자 인터페이스는 기본적으로 데이터 모듈의 메소드들을 참조한다. 사용자 인터페이스는 하나의 메인 폼을 사용하며, 메인 폼 안에는 세 개의 페이지가 들어간다. 첫 번째 페이지는 버그 정보를 추가, 수정, 조회하기 위한 것이다. 두 번째 페이지는 주석들을 조회하기 위한 것이다. 세 번째 페이지는 그리드를 통해서 모든 버그들 또는 현재 로그인한 사용자가 보고한 버그들만을 보여준다. 그림 32.2, 그림 32.3, 그림 32.4는 메인 폼의 세 페이지들을 나타낸 것이다.

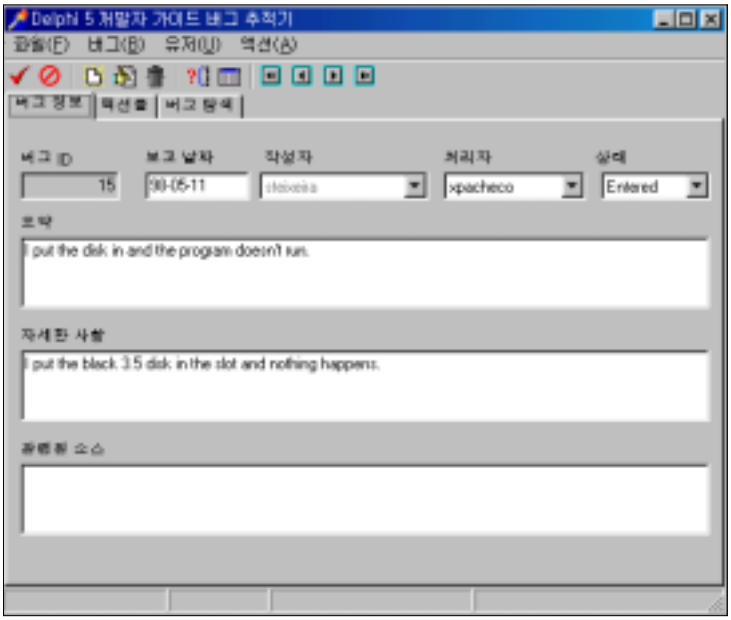


그림 32.2
버그 정보 페이지

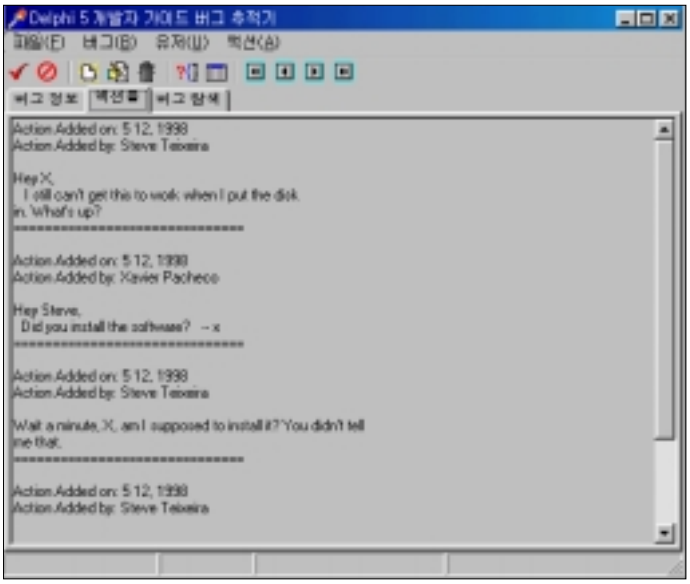


그림 32.3
주석 페이지

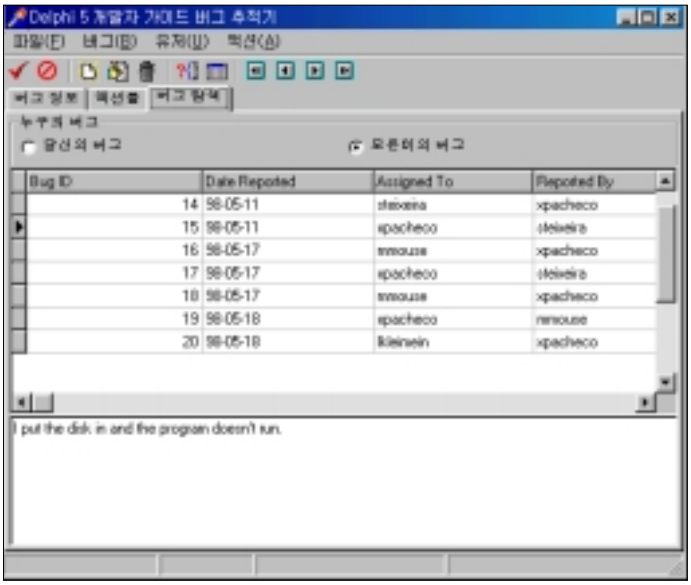


그림 32.4
버그 조회 페이지

목록 32.5 Ch32\MainForm

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, DBNAVSTATFRM, Menus, ImgList, ComCtrls, ToolWin, StdCtrls,
  DBCtrls, Db, Mask, dbModeFrm, ActnList, Grids, DBGrids, ExtCtrls;

type

  TMainForm = class(TDBNavStatForm)
    pcMain: TPageControl;
    tsBugInformation: TTabSheet;
    tsActions: TTabSheet;
    lblBugID: TLabel;
    dbeBugID: TDBEdit;
    dsBugs: TDataSource;
    lblDateReported: TLabel;
    lblSummary: TLabel;
    lblDetails: TLabel;
    lblAffectedSource: TLabel;
    lblReportedBy: TLabel;
    lblAssignedTo: TLabel;
    lblStatus: TLabel;
    dbmSummary: TDBMemo;
    dbmDetails: TDBMemo;
    dbmAffectedSource: TDBMemo;
    tsBrowseBugs: TTabSheet;
    rgWhoseBugs: TRadioGroup;
    dbgBugs: TDBGrid;
    dbmSummary2: TDBMemo;
    memAction: TMemo;
    dblcAssignedTo: TDBLookupComboBox;
    dblcStatus: TDBLookupComboBox;
```

```
    dbeDateReported: TDBEdit;
    mmiUsers: TMenuItem;
    mmiAddUser: TMenuItem;
    mmiActions: TMenuItem;
    mmiAddActionToBug: TMenuItem;
    dblcReportedBy: TDBLookupComboBox;
    Exit1: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure sbFirstClick(Sender: TObject);
    procedure sbPreviousClick(Sender: TObject);
    procedure sbNextClick(Sender: TObject);
    procedure sbLastClick(Sender: TObject);
    procedure sbSearchClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure sbAcceptClick(Sender: TObject);
    procedure sbCancelClick(Sender: TObject);
    procedure sbInsertClick(Sender: TObject);
    procedure sbEditClick(Sender: TObject);
    procedure sbDeleteClick(Sender: TObject);
    procedure sbBrowseClick(Sender: TObject);
    procedure rgWhoseBugsClick(Sender: TObject);
    procedure mmiExitClick(Sender: TObject);
    procedure mmiAddUserClick(Sender: TObject);
    procedure mmiAddActionToBugClick(Sender: TObject);
    procedure dsBugsDataChange(Sender: TObject; Field: TField);
  private
    procedure SetActionStatus;

  protected
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation
```

```
uses DDGBugsDM;

{$R *.DFM}

{ TMainForm }

procedure TMainForm.SetActionStatus;
begin
  mmiFirst.Enabled := not DDGBugsDataModule.IsFirstBug;
  mmiLast.Enabled  := not DDGBugsDataModule.IsLastBug;
  mmiNext.Enabled  := not DDGBugsDataModule.IsLastBug;
  mmiPrevious.Enabled := not DDGBugsDataModule.IsFirstBug;
  mmiDelete.Enabled := not DDGBugsDataModule.IsBugsTblEmpty;

  sbFirst.Enabled := mmiFirst.Enabled;
  sbLast.Enabled  := mmiLast.Enabled;
  sbNext.Enabled  := mmiNext.Enabled;
  sbPrev.Enabled  := mmiPrevious.Enabled;
  sbDelete.Enabled := mmiDelete.Enabled;

  // 버그를 추가/수정하는 도중에는 사용자나 주석을 추가하지 못하게 한다.
  mmiUsers.Enabled := FormMode = fmBrowse;
  mmiActions.Enabled := (FormMode = fmBrowse) and
      (DDGBugsDataModule.NumBugs <> 0);

 { 사용자가 버그를 편집하거나 추가할 때에는 버그 레코드들을 조회하지 못하게
   한다. }
  dbgBugs.Enabled     := FormMode = fmBrowse;
  rgWhoseBugs.Enabled := FormMOde = fmBrowse;

end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  inherited;
  SetActionStatus;
end;
```

```
procedure TMainForm.sbFirstClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.FirstBug;
  SetActionStatus;
end;

procedure TMainForm.sbPreviousClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.PreviousBug;
  SetActionStatus;
end;

procedure TMainForm.sbNextClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.NextBug;
  SetActionStatus;
end;

procedure TMainForm.sbLastClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.LastBug;
  SetActionStatus;
end;

procedure TMainForm.sbSearchClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.SearchForBug;
end;

procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose:
 Boolean);
var
```

```
  Rslt: word;
begin
  inherited;
  if not (FormMode = fmBrowse) then
  begin
    rslt := MessageDlg('Save changes?', mtConfirmation, mbYesNoCancel, 0);
    case rslt of
      mrYes:
        begin
          DDGBugsDataModule.SaveBug;
          FormMode := fmBrowse;
          CanClose := True;
        end;
      mrNo:
        begin
          DDGBugsDataModule.CancelBug;
          FormMode := fmBrowse;
          CanClose := True;
        end;
      mrCancel:
        CanClose := False;
    end;
  end;
end;

procedure TMainForm.sbAcceptClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.SaveBug;
  SetActionStatus;
end;

procedure TMainForm.sbCancelClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.CancelBug;
  SetActionStatus;
end;
```

```
procedure TMainForm.sbInsertClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.InsertBug;
  SetActionStatus;
end;

procedure TMainForm.sbEditClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.EditBug;
  SetActionStatus;
end;

procedure TMainForm.sbDeleteClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.DeleteBug;
  SetActionStatus;
end;

procedure TMainForm.sbBrowseClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.CancelBug;
  SetActionStatus;
end;

procedure TMainForm.rgWhoseBugsClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.FilterOnUser := rgWhoseBugs.ItemIndex = 0;
end;

procedure TMainForm.mmiExitClick(Sender: TObject);
begin
  inherited;
```

```
  Close;
end;

procedure TMainForm.mmiAddUserClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.AddUser;
end;

procedure TMainForm.mmiAddActionToBugClick(Sender: TObject);
begin
  inherited;
  DDGBugsDataModule.AddAction;
  dsBugsDataChange(nil, nil);
end;

procedure TMainForm.dsBugsDataChange(Sender: TObject; Field: TField);
begin
  inherited;
 // 새 버그 레코드가 표시되는 중이므로 주석 목록을 비우고
 // 표시되고 있는 버그에 대한 주석 레코드들을 가져온다.
  memAction.Lines.Clear;
  DDGBugsDataModule.GetActions(memAction.Lines);
end;

 end.
```

TMainForm은 4장, "애플리케이션 프레임웍과 설계 개념들"에서 만들었던 TDBNavStatForm을 상속한다. TDBNavStatForm에는 폼의 상태(추가, 수정, 조회)에 따라 스피드 버튼들을 갱신하는 기능이 들어 있다. TMainForm의 대부분의 메소드들은 그냥 데이터 모듈의 해당 메소드들을 호출하는 역할만을 수행한다.

그리고 Bugs 테이블이 잘못해서 수정 모드로 들어가는 것을 막기 위해 dsBug.AutoEdit 속성을 False로 했다는 점을 주목하기 바란다. 사용자가 적절한 스피드 버튼을 직접 클릭한 경우에만 Bugs 테이블이 수정이나 추가 모드로 설정되도록 하기 위한 것이다.

SetActionStatus()는 폼의 상태에 따라 버튼들과 메뉴 항목들을 적절히 활성화/비활성화시킨다. FormCloseQuery()는 폼이 종료되기 전에 아직 저장되지 않은 변경을 저장하거나 취소할 수 있는 기회를 제공한다.

사용자 인터페이스에 관련된 기타 사항

TDBGrid의 각 컬럼 제목은 기본적으로 각 컬럼에 추가된 해당 필드 이름과 같게 되는데, 필드 이름들을 사용자에게 직접 표시하는 것은 별로 좋지 않다. 그래서 TDBGrid.Columns의 Title 속성을 통해서 좀더 의미 있고 친근한 제목을 설정했다. 폼의 레이블 컴포넌트들 역시 이해하기 쉬운 텍스트들로 설정했다.

웹과의 연동 준비 작업

다음 장에서는 이 애플리케이션을 웹에서 사용할 수 있도록 개선시킬 것이다. 그렇게 하기 위해서는 TDDGBugsDataModule 안에서 어떤 폼을 참조하는 부분이 있어서는 안 된다. DDGBugsDM.pas의 소스 코드 안에 가끔 나타나는 조건부 컴파일 지시자는 그런 목적을 만족시키기 위한 것들이다. 그 예로, 목록 32.1에 다음과 같은 코드가 있다.

```
{$IFNDEF DDGWEBBUGS}
procedure AddUser;
{$ENDIF}
```

AddUser() 메소드는 DDGWEBBUGS가 정의되어 있지 않은 경우에만 컴파일된다(이번 버전이 그렇다). 웹 버전의 애플리케이션에서는 DDGWEBBUGS가 정의되어 있다.

요약

이번 장에서는 데스크탑 데이터베이스 애플리케이션을 개발하는 기법들에 대해서 설명했다. 특히, 데이터 조작 부분과 사용자 인터페이스 부분을 철저하게 분리시키는 것에 중점을 두었다. 두 부분이 분리되어 있으면 나중에 웹 버전으로 전환하는 작업이 매우 쉬워진다. 다음 장에서는 실제로 이 애플리케이션을 웹 버전으로 전환해 보겠다.