33장 DDG 버그 보고용 도구 - 웹 브로커 활용

* 페이지 구성
* 데이터 모듈의 수정
* TDataSetTableProducer 컴포넌트 dstpBugs의 설정
* TWebDispatcher 컴포넌트 wbdpBugs의 설정
* TPageProducer 컴포넌트 pprdBugs의 설정
* DDGWebBugs ISAPI 서버의 작성 - TWebActionItem들을 추가
* 버그 조회
* 새 버그 정보 추가

32장, "DDG 버그 보고용 도구 - 데스크탑 애플리케이션 개발"에서는 데스크탑 데이터베이스 애플리케이션 작성에 관한 여러 가지 기법들을 설명했었다. 32장에서 중점적으로 고민했던 것들 중 하나는 데스크탑 데이터베이스 애플리케이션을 웹에 쉽게 적용할 수 있는 구조로 만드는 것이었다. 이번 장에서는 32장에서 만든 버그 보고용 애플리케이션을 ISAPI 서버로 전환해서 웹에 적용시키겠다. 32장에서 데이터 조작 부분과 사용자 인터페이스 부분을 완전히 분리시켰었기 때문에 최소한의 코딩만으로도 쉽게 애플리케이션을 웹 버전으로 전환시킬 수 있다. 웹 애플리케이션을 만드는 기법들에 대해서는 30장, "웹 브로커를 이용한 웹 애플리케이션 작성"에서 설명했으므로 이번 장에서는 동일한 내용을 반복하지 않겠다. 웹 애플리케이션 작성에 대해 아직 잘 이해가 가지 않는 독자는 30장을 다시 읽어보는 것도 좋을 것이다.

페이지 구성

웹 기반 버그 보고용 도구의 흐름 또는 구성은 그림 33.1과 같다.

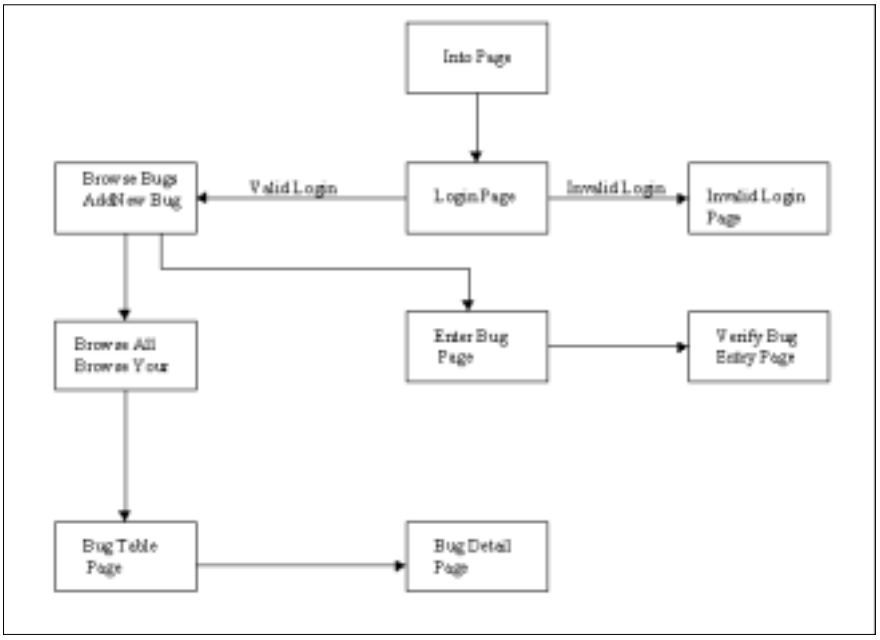


그림 33.1
웹 기반 버그 보고용 도구의 흐름

페이지 구성을 보면 알겠지만, 이번 장의 웹 기반 버그 보고용 도구는 32장에 나오는 데스크탑 기반 애플리케이션의 사용자 인터페이스 부분을 웹 페이지들로 대체한 것이라고 생각하면 된다. 데이터 조작 부분은 데이터 모듈에 집중되어 있으므로 크게 손 볼 것은 없다.

각각의 페이지에 대해서는 잠시 후에 개별적으로 설명하겠다. 그런데 위에 나온 모든 페이지들은 모두 동적으로 생성되는 것들이다. 즉, 디스크상의 HTML 파일을 직접 보여주는 경우는 없으며, 모든 웹 페이지들은 실행 시점에서 코드를 통해서 생성된다. 물론, 반드시 그렇게 해야 하는 것은 아니다. 좀더 멋진 웹 페이지들을 보여주고 싶다면 HTML 템플릿과 웹 브로커 컴포넌트들을 적절히 조합하는 것도 좋을 것이다.

데이터 모듈의 수정

33장과 32장의 주요 목적이라면 데이터 조작 기능을 모두 가지고 있는 하나의 데이터 모듈을 서로 다른 상황의 애플리케이션에서 재사용할 수 있는 기법을 보여 주는 것이다. 이번 웹 기반 애플리케이션에서 사용할 데이터 모듈은 32장에서 만들었던 데이터 모듈에 웹 관련 기능들을 추가한 것이다. 기존의 메소드들은 건드리지 않았으므로 이번 장에서 수정한 데이터 모듈을 다시 저번 장의 애플리케이션에서 그대로 포함시켜서 컴파일해도 문제가 생기지 않는다.

그리고 이번 장의 애플리케이션에는 개별적인 웹 모듈을 사용하지 않았다. 그냥 기존의 TDataModule에 TWebDispatcher 컴포넌트를 추가했을 뿐이다. 이렇게 하면 기존에 만들어 두었던 TDataModule을 웹 애플리케이션에 그대로 사용할 수 있게 된다.

이번 웹 기반의 버그 보고용 도구를 위해서 TDDGBugsDataModule에는 네 개의 컴포넌트들을 추가했다. 추가된 컴포넌트들은 TWebDispatcher, TDataSetTableProducer, TPageProducer, TSession이다. 이 컴포넌트들은 대부분 웹 페이지를 동적으로 생성해 내는 역할들을 담당한다.

TSession 컴포넌트는 웹 페이지의 생성에 직접 관련이 있는 것은 아니다. 웹 애플리케이션은 불특정 다수가 사용하게 되므로 하나의 DLL 인스턴스(ISAPI 서버)를 통해서 여러 클라이언트가 동시에 데이터베이스에 접근하는 상황이 벌어질 수 있다. 하나의 DLL 프로세스 안에서 각각의 클라이언트마다 개별적인 웹 모듈 인스턴스가 생

성되며, 각각의 웹 모듈 인스턴스는 개별적인 스레드에서 실행된다. 개별적인 웹 모듈의 데이터베이스 조작이 각각 독립적으로 수행될 수 있도록 하는 것이 바로 TSession 컴포넌트이다. TSession 컴포넌트의 AutoSessionName 속성을 True로 설정하면 각각의 웹 모듈에 생성된 TSession 인스턴스는 고유한 이름을 가지므로 동시적인 데이터베이스 접근 때문에 생기는 충돌을 피할 수 있다.

그런데 WinCGI나 CGI 확장 기능 같은 경우에는 각각의 클라이언트마다 개별적인 프로세스에서 실행되므로 웹 모듈이나 TDataModule에 TSession 컴포넌트를 추가할 필요가 없다는 점도 기억해 두기 바란다.

TDataSetTableProducer 컴포넌트 dstpBugs의 설정

데이터 모듈의 TDataSetTableProducer 컴포넌트인 dstpBugs는 TTable 컴포넌트 tblBugs에 연결되어 있다. 사용자의 이해를 돕기 위해 dstpBugs.Columns 속성을 수정해서 컬럼 제목들을 좀더 의미있는 이름으로 만들었다(그림 33.2). 컬럼 제목들은 웹 페이지의 HTML 테이블에 나타난다. 또한, dstpBugs.TableAttributes 속성을 통해서 테이블의 경계선을 1픽셀 너비로 설정했다. 그렇게 하면 대부분의 웹 브라우저들은 테이블을 입체적인 느낌이 나도록 표시한다.

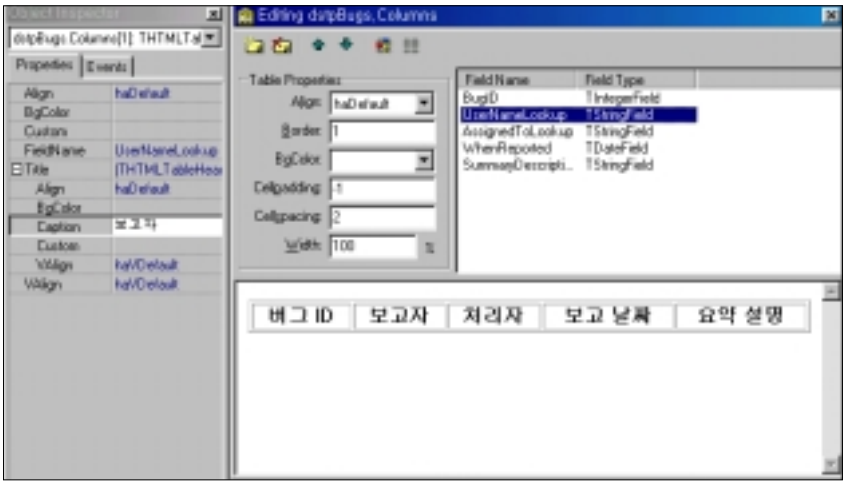


그림 33.2
dstpBugs의 Columns 속성을 편집

TWebDispatcher 컴포넌트 wbdpBugs의 설정

그림 33.3은 Actions 속성 편집기를 이용해서 wbdpBugs에 TWebActionItem들을 추가한 모습이다. 잠시 후에 코드를 통해서 각각의 액션들에 대해 설명하겠다.

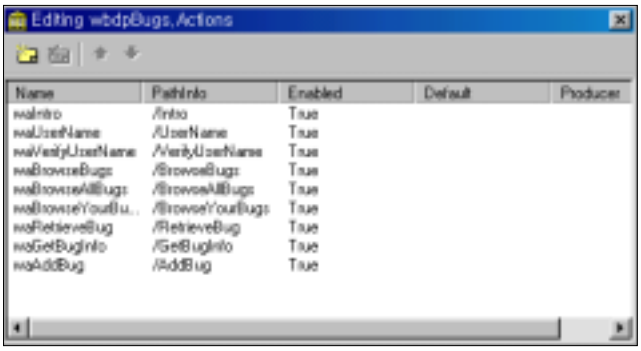


그림 33.3
wbdpBugs의 Actions 속성을 편집

TPageProducer 컴포넌트 pprdBugs의 설정

pprdBugs.HTMLDoc 속성에는 아무 것도 설정하지 않는다. 이 속성은 실행 시점에서 코드를 통해서 채워진다. TPageProducer는 서로 다른 두 상황에서 사용하는데, 이에 대해서는 나중에 코드 부분에서 설명하겠다.

DDGWebBugs ISAPI 서버의 작성 - TWebActionItem들을 추가

웹 기반 버그 보고용 도구의 모든 기능은 TWebDispatcher 컴포넌트의 TWebActionItem들을 통해서 제공된다. 각 TWebActionItem 인스턴스들의 목적은 표 33.1에 나와 있다. 각각의 TWebActionItem들에 대해서는 잠시 후에 개별적으로 설명하겠다.

표 33.1 TWebActionItem 인스턴스들의 목적

TWebActionItem	목적
waIntro	사용자에게 시작 페이지를 표시한다.
waUserName	사용자 이름(username)을 입력받는다.
waVerifyUserName	waUserName.OnAction에서 호출된다. 사용자가 입력한 사용자 이름을 확인한다.
waBrowseBugs	사용자에게 전체 버그들을 볼 것인지, 또는 사용자 자신이 보고한 버그들만 볼 것인지를 묻는다.
waBrowseAllBugs	데이터베이스의 모든 버그들을 포함하는 테이블을 표시한다.
waBrowseYourBugs	사용자가 보고한 버그들만을 포함하는 테이블을 표시한다.
waRetrieveBug	버그에 대한 상세 정보를 표시한다.
waGetBugInfo	새 버그 정보를 입력할 수 있는 페이지를 표시한다.
waAddBug	테이블에 새 버그를 추가하고 확인 페이지를 표시한다.

그럼 이제 코드를 살펴보자. 저번 장의 DDGBugsDM.pas에 추가된 부분만을 설명하겠다.

범용 루틴들

목록 33.1의 AddHeader() 프로시저는 웹 기반 버그 보고용 도구에서 공통적으로 사용하는 페이지 제목과 머리말을 추가하기 위한 것이다. 웹 페이지의 배경 이미지도 이 프로시저에서 설정된다. 배경 이미지의 경로는 웹 서버마다 다를 것이므로 독자가 이번 예제를 실행할 때에는 /samples/images/ 부분을 적절히 수정해야 할 것이다. 목록 33.2에 나온 AddFooter()는 공통적으로 사용하는 꼬리말을 추가하기 위한 것이다. 꼬리말에는 저작권 정보가 포함된다.

목록 33.1 TDDGBugsDataModule.AddHeader()

```
procedure AddHeader(AWebPage: TStringList);
// 각 웹 페이지에 공통적인 머리말을 추가한다.
begin
  with AWebPage do
  begin
    Add('<HTML>');
    Add('<HEAD>');
    Add('<BODY BACKGROUND="/samples/images/backgrnd.gif">');
    Add('<TITLE>Delphi 5 개발자 가이드 버그 데모</Title>');
    Add('<CENTER>');
    Add('<P>');
    Add('<FONT SIZE=6>Delphi 5 개발자 가이드 버그 데모</font>');
    Add('</CENTER>');
    Add('</HEAD>');
  end;
  end;
```

목록 33.2 TDDGBugsDataModule.AddFooter()

```
procedure AddFooter(AWebPage: TStringList);
// 각 웹 페이지에 공통적인 꼬리말을 추가한다.
begin
  with AWebPage do
  begin
    Add('<BR><BR>저작권 (c) 1999, Delphi 5 개발자 가이드.');
    Add('</BODY>');
    Add('</HTML>');
  end;
  end;
```

시작 페이지

그림 33.4는 시작 페이지의 모습이다. 이 페이지는 waIntro.OnAction 이벤트 처리 메소드인 wbdpBugswaIntroAction()이 만들어낸다(목록 33.3).

목록 33.3 TDDGBugsDataModule.wbdpBugswaIntroAction()

```
procedure TDDGBugsDataModule.wbdpBugswaIntroAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
// 시작 페이지를 생성한다.
var
  WebPage: TStringList;
begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    with WebPage do
    begin
      Add('<BODY>');
      Add('<H1>소개</H1>');
      Add('<P>델파이 5 개발자 가이드 버그 데모에 오신 것을 환영합니다.');
      Add('<BR이 데모는 현존하는 애플리케이션을 웹 애플리케이션과 어떻게
      연동시켜 주는지 보여주는 데모입니다.');
      Add('<BR>데모를 테스트: 버그를 추가하기 위해 또는 버그를 탐색하기 위해
      로그온 링크와 다음 페이지를 클릭하세요.');
      Add('</P>');
      Add('<A href="../DDGWebBugs.dll/UserName">DDG 버그 데모에 로그온</A>');
      AddFooter(WebPage);
      Response.Content :=  WebPage.Text;
      Handled := True;
    end;
  finally
    WebPage.Free;
  end;
end;
```

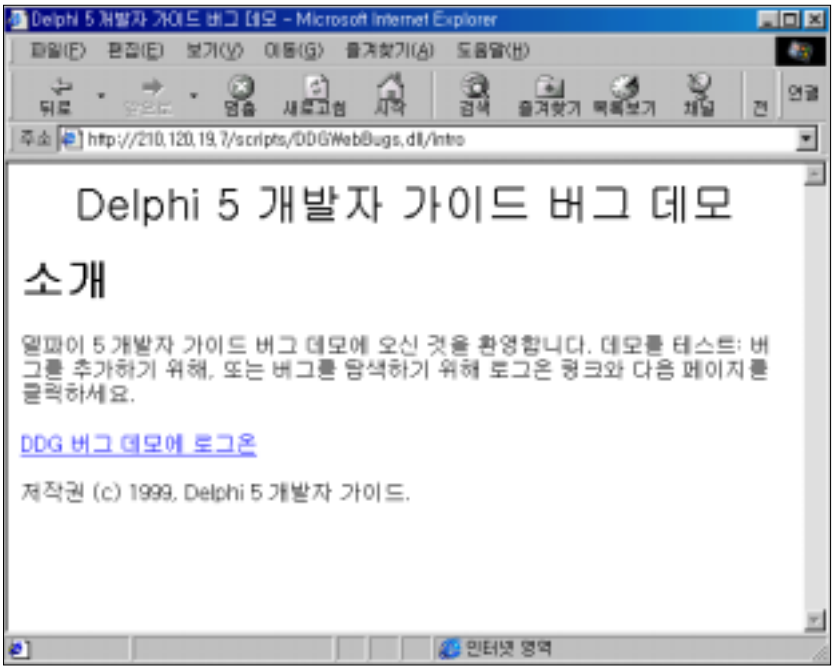


그림 33.4
시작 페이지

각각의 웹 페이지에는 AddHeader()와 AddFooter()를 통해서 공통적인 제목과 머리말, 꼬리말이 추가된다. 시작 페이지에는 TWebActionItem 인스턴스인 waUserName을 가리키는 링크가 포함되어 있다.

사용자 로그인 페이지와 사용자 이름 확인 페이지

그림 33.5는 TDDGBugsDataModule.wbdpBugswaUserNameAction()(목록 33.4)이 만들어 내는 페이지이다. 이 페이지는 HTML 양식(form)을 통해서 사용자 이름을 입력받는다. Submit 버튼을 클릭하면 TDDGBugsDataModule.wbdpBugswaVerifyUseNameAction() 이벤트 처리 메소드가 호출된다(목록 33.5).

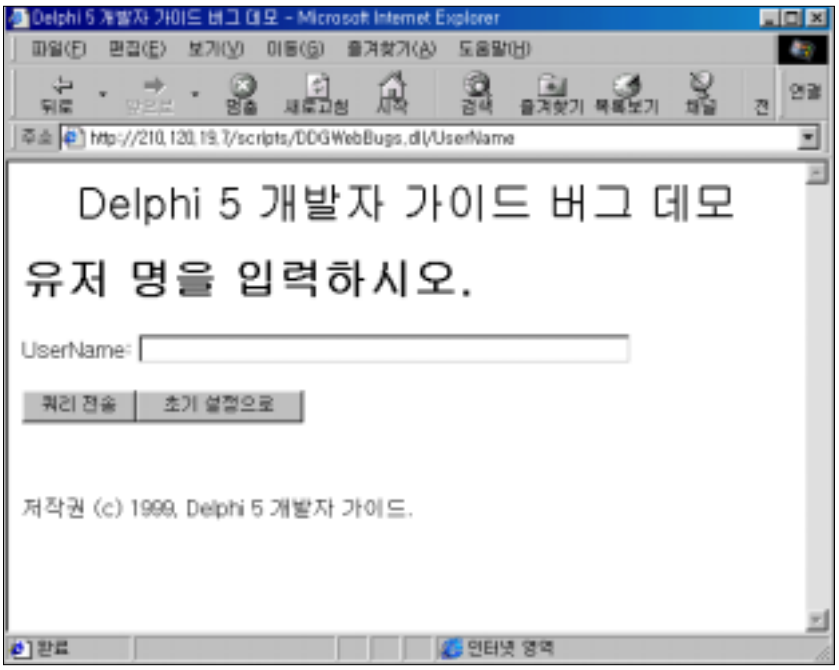


그림 33.5
사용자 이름을 입력받는 페이지

목록 33.4 TDDGBugsDataModule.wbdpBugswaUserNameAction()

```
procedure TDDGBugsDataModule.wbdpBugswaUserNameAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
// 사용자 이름을 입력받는 페이지를 생성한다.
var
  WebPage: TStringList;
begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    with WebPage do
    begin
      Add('<BODY>');
      Add('<H1>유저명을 입력하시오.</H1>');
```

```
      Add('<FORM action="../DDGWebBugs.dll/VerifyUserName" ' +
              'method="GET">');
      Add('<p>UserName: <INPUT type="text" name="UserName" ' +
              'maxlength="30" size="50"></P>');
      Add('<p><INPUT type="SUBMIT"><INPUT type="RESET"></p>');
      Add('</FORM>');
      AddFooter(WebPage);
      Response.Content :=  WebPage.Text;
      Handled := True;
    end;
  finally
    WebPage.Free;
  end;
end;
```



TDDGBugsDataModule.wbdpBugswaVerifyUserNameAction()

```
procedure TDDGBugsDataModule.wbdpBugswaVerifyUserNameAction(
  Sender: TObject; Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
{ 이 페이지는 사용자가 입력한 사용자 이름을 넘겨 받는다. 그리고 사용자 이름을
  포함한 일련의 정보를 쿠키에 담아서 클라이언트에게 보낸다. 나중에 사용자가 버그
  정보를 추가할 때 쿠키에 기록된 정보를 이용한다. }

var
  WebPage: TStringList;
  CookieList: TStringList;
  UserName: String;
  UserFName,
  UserLName: String;
  UserID: Integer;
  ValidLogin: Boolean;

procedure BuildValidLoginPage;
```

```
begin
  AddHeader(WebPage);
  with WebPage do
  begin
    Add('<BODY>');
    Add(Format('<H1>유저명, %s 검증됨. 유저 ID는: %d</H1>',
      [Request.QueryFields.Values['UserName'], UserID]));
    Add('<BR><BR><A href="../DDGWebBugs.dll/BrowseBugs">버그 리스트
탐색</A>');
    Add('<BR><A href="../DDGWebBugs.dll/GetBugInfo">새 버그 추가</A>');
    AddFooter(WebPage);
  end;
end;

procedure BuildInValidLoginPage;
begin
  AddHeader(WebPage);
  with WebPage do
  begin
    Add('<BODY>');
    Add(Format('<H1>유저 명, %s는 잘못된 유저입니다.</H1>',
      [Request.QueryFields.Values['UserName']]));
    AddFooter(WebPage);
  end;
end;

begin

  UserName := Request.QueryFields.Values['UserName'];

 // 사용자 이름이 Users.db에 존재하면 유효한 로그인으로 간주한다.
  ValidLogin := tblUsers.Locate('UserName', UserName, []);

  WebPage := TStringList.Create;
  try

    if ValidLogin then
    begin
```

```
    // 사용자의 ID와 이름, 성을 얻는다.
    UserID := tblUsers.FieldByName('UserID').AsInteger;
    UserFName := tblUsers.FieldByName('UserFirstName').AsString;
    UserLName := tblUsers.FieldByName('UserLastName').AsString;

    CookieList := TSTringList.Create;
    try

      // 사용자의 정보를 쿠키로 저장한다.
      CookieList.Add('UserID='+IntToStr(UserID));
      CookieList.Add('UserName='+UserName);
      CookieList.Add('UserFirstName='+UserFName);
      CookieList.Add('UserLastName='+UserLName);

      Response.SetCookieField(CookieList, '', '', Now + 1, False);
    finally
      CookieList.Free;
    end;
    BuildValidLoginPage;
  end
  else begin
    UserID := -1;
    BuildInvalidLoginPage;
  end;

  Response.Content :=  WebPage.Text;
  Handled := True;

 finally
  WebPage.Free;
 end;

end;
```

wbdpBugswaVerifyUserNameAction()은 우선 입력된 사용자 이름이 tblUsers 테이블에 존재하는지 확인한다. 사용자 이름이 존재하면 BuildValidLoginPage() 프로시저를 호출하고, 존재하지 않으면 BuildInvalidLoginPage()를 호출한다.

유효한 로그인인 경우에는 tblUsers에서 사용자의 이름과 성 그리고 사용자 ID를 얻고 그것들을 쿠키로 만들어서 클라이언트에게 보낸다. 이후의 웹 페이지들은 쿠키에 기록된 정보를 통해서 사용자를 식별하게 된다. 쿠키를 기록한 다음에는 BuildVaild LoginPage()를 호출하는데, BuildVaildLoginPage()가 만들어내는 페이지에는 버그 조회나 입력 페이지로 가는 링크가 포함되어 있다. 잘못된 로그인인 경우에는 BrowseInvalidLoginPage()가 수행되는데, 이 프로시저는 사용자 이름이 잘못되었음을 알리는 페이지를 표시한다.

유효한 사용자 이름을 입력한 사용자는 버그 조회나 새 버그 입력 중 하나를 선택하게 된다.

버그 조회

사용자가 버그 조회를 선택한 경우에는 데이터베이스의 모든 버그들을 볼 것인지 아니면 사용자 자신이 입력한 버그들만을 볼 것인지를 선택하는 페이지가 나타나는데, 그 페이지를 만들어내는 것이 목록 33.6에 나온 TDDGBugsDataModule.wbdpBugswaBrowseBugsAction() 메소드이다.

목록 33.6 TDDGBugsDataModule.wbdpBugswaBrowseBugsAction()

```
procedure TDDGBugsDataModule.wbdpBugswaBrowseBugsAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
{ 데이터베이스의 모든 버그들을 볼 것인지 아니면 사용자 자신이 입력한 버그들만을
  볼 것인지를 선택하는 페이지를 만들어낸다. }
var
  WebPage: TStringList;
begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    with WebPage do
```

```
    begin
      Add('<BODY>');
      Add('<H1>탐색 옵션</H1>');
      Add('<BR><BR><A href="../DDGWebBugs.dll/BrowseAllBugs">' +
              '모든 버그 탐색하기</A>');
      Add('<BR><A href="../DDGWebBugs.dll/BrowseYourBugs">' +
              '당신의 버그 탐색하기</A>');
      AddFooter(WebPage);
      Response.Content :=  WebPage.Text;
      Handled := True;
    end;
  finally
    WebPage.Free;
  end;

end;
```

모든 버그들을 조회

사용자가 모든 버그들을 보겠다고 하면 TDDGBugsDataModule.wbdpBugswaBrowseAllBugsAction() 이벤트 처리 메소드가 실행된다(목록 33.7).

목록 33.7 TDDGBugsDataModule.wbdpBugswaBrowseAllBugsAction()

```
procedure TDDGBugsDataModule.wbdpBugswaBrowseAllBugsAction(Sender:
TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
{ 공통 머리말과 꼬리말 사이에 <#TABLE>이라는 태그가 들어간 페이지를 만든다.
  그리고 TPageProducer 컴포넌트를 통해서 #TABLE 태그를 모든 버그들이 포함된
  HTML 테이블로 치환한다. }
var
  WebPage: TStringList;
begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    WebPage.Add('<BODY>');
```

```
    WebPage.Add('<H1>모든 버그들 탐색하기</H1>');
    WebPage.Add('<#TABLE>');
    AddFooter(WebPage);

    pprdBugs.HTMLDoc.Clear;
    pprdBugs.HTMLDoc.AddStrings(WebPage);

      { 다음 행을 통해서 pprdBugs의 OnHTMLTag 이벤트 처리 메소드가 수행된다. }
    Response.Content := pprdBugs.Content;

    Handled := True;
  finally
    WebPage.Free;
  end;
end;
```

이 이벤트 처리 메소드는 TPageProducer 컴포넌트인 pprdBugs를 사용한다. 이 메소드는 공통적인 머리말과 꼬리말 사이에 #TABLE이라는 태그를 포함한 HTML 페이지를 만든 다음, 그것을 Response.Content 속성에 직접 넣는 대신 pprdBugs.HTMLDoc 속성에 넣는다. 그리고 나서 pprdBugs.Content를 Response.Content에 넣으면 pprdBugs.OnHTML Tage 이벤트가 발생한다. 이 이벤트를 처리하는 메소드는 목록 33.8에 나온 TDDGBugsDataModule.pprdBugsHTMLTag()이다.

목록 33.8 TDDGBugsDataModule.pprdBugsHTMLTag()

```
procedure TDDGBugsDataModule.pprdBugsHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  if Tag = tgTable then begin
    with dstpBugs do
    begin
      DataSet.Open;
      ReplaceText := dstpBugs.Content;
      DataSet.Close;
    end;
  end;
end;
```

이 이벤트 처리 메소드는 버그 테이블을 가리키는 dstpBugs.Content 속성을 pprdBugs.ReplaceText 속성에 넣는다. 그렇게 하면 #TABLE 태그가 HTML 테이블로 치환된다. 그림 33.6은 그 결과로 생성된 HTML 페이지이다. 이 페이지에는 모든 사용자들이 입력한 버그들이 표시된다.

사용자가 입력한 버그들만을 조회

사용자가 자신이 입력한 버그들만을 보겠다고 선택한 경우에는 그 사용자가 입력한 버그들로만 된 테이블을 포함하는 페이지가 나타난다. 그 페이지는 목록 33.9에 나온 TDDGBugsDataModule.wbdpBugswaBrowseYourBugsAction() 이벤트 처리 메소드가 만들어낸다.

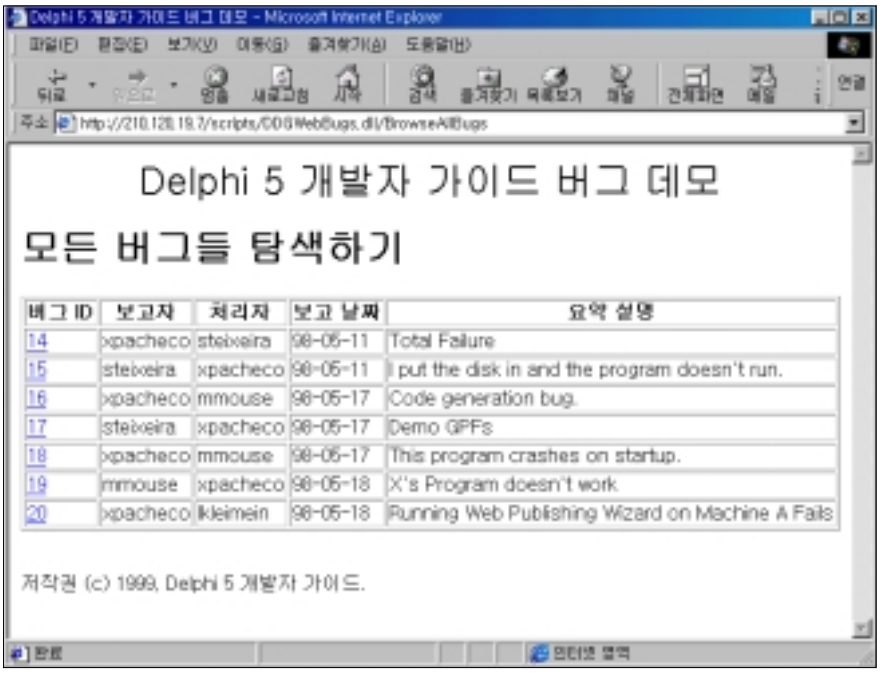


그림 33.6
모든 사용자들이 입력한 버그들의 목록

목록 33.9 TDDGBugsDataModule.wbdpBugswaBrowseYourBugsAction()

```
procedure TDDGBugsDataModule.wbdpBugswaBrowseYourBugsAction(
  Sender: TObject; Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
{ 공통 머리말과 꼬리말 사이에 <#TABLE>이라는 태그가 들어간 페이지를 만든다.
  그리고 TPageProducer 컴포넌트를 통해서 #TABLE 태그를 사용자가 입력한
  버그들이 포함된 HTML 테이블로 치환한다. }
var
  WebPage: TStringList;
  UserID: Integer;
  UserFName,
  UserLName: String;
begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    WebPage.Add('<BODY>');

    // 쿠키에 저장되어 있는 사용자 ID를 얻는다.
    UserID    := StrToInt(Request.CookieFields.Values['UserID']);
    UserFName := Request.CookieFields.Values['UserFirstName'];
    UserLName := Request.CookieFields.Values['UserLastName'];

    WebPage.Add(Format('<H1> %s %s에 입력된 버그 탐색하기</H1>',
      [UserFName, UserLName]));
    WebPage.Add('<#TABLE>');
    pprdBugs.HTMLDoc.Clear;
    pprdBugs.HTMLDoc.AddStrings(WebPage);

    AddFooter(WebPage);

    // 테이블을 사용자 ID를 통해서 걸러낸다.
    FLoginUserID := UserID;
    FilterOnUser := True;

    Response.Content := pprdBugs.Content;
```

```
    Handled := True;
  finally
    WebPage.Free;
  end;

end;
```

모든 버그들을 보여주는 메소드와 마찬가지로 이 메소드 역시 #TABLE 태그와 pprdBugs를 이용해서 HTML 테이블을 생성한다. 이 메소드는 Request.CookieFields 속성을 통해서 쿠키에 저장된 사용자 ID와 사용자의 이름, 성을 얻는다. 사용자 이름과 성은 웹 페이지에 사용자의 전체 이름을 표시하는 데 사용한다. 그리고 사용자 ID(UserID 변수)를 FLoginUserID에 넣은 다음, FilterOnUser 속성을 True로 설정한다. 32장에서 FilterOnUser 속성을 True로 하면 FilterOnUser 속성의 쓰기 메소드인 SetFilterOnUser()가 수행되도록 했었다. SetFilterOnUser()는 tblBugs.Filtered를 True로 하며, 결과적으로 tblBugs의 각각의 레코드에 대해 OnFilterRecord 이벤트가 발생한다. OnFilterRecord 이벤트를 처리하는 tblBugsFilterRecord() 메소드는 다음과 같은 코드를 수행한다.

```
Accept := tblBugs.FieldByName('UserID').AsInteger = FLoginUserID;
```

위의 문장의 의미는 FLoginUserID와 같은 값을 가지는 UserID 필드가 있는 레코드들만 통과시킨다는 뜻이다. 쿠키로부터 얻은 UserID를 FLoginUserID에 넣는 것은 바로 이 코드 때문인 것이다.

pprdBugs.Content 속성을 Response.Content에 넣으면 pprdBugs.OnHTMLTag 이벤트가 발생하며, 결과적으로 #TABLE 태그가 HTML 테이블로 치환된다.

테이블 셋의 포매팅과 버그 상세 정보 표시

테이블의 각 셀을 포매팅하는 것은 dstpBugs의 OnFormatCell 이벤트 처리 메소드인 TDDGBugsDataModule.dstpBugsFormatCell()이다. 이 이벤트 처리 메소드는 BugID를 HTML 링크로 변환한다. 그 링크가 가리키는 것은 해당 버그에 대한 상세 정보 페이지이다. 상세 정보 페이지를 실제로 만들어내는 것은 wbdpBugswaRetrieveBugAction() 이벤트 처리 메소드이다. 두 메소드는 목록 33.10에 나와 있다.

목록 33.10 버그 상세 정보를 표시하기 위한 이벤트 처리 메소드들

```
procedure TDDGBugsDataModule.dstpBugsFormatCell(Sender: TObject; CellRow,
  CellColumn: Integer; var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
{ 테이블의 BugID 셀을 버그 상세 정보 페이지로 가기 위한 링크로 변환한다. }
begin
  if (CellColumn = 0) and not (CellRow = 0) then
    CellData := Format('<A href="../DDGWebBugs.dll/RetrieveBug?' +
          'BugID=%s">%s</A>', [CellData, CellData]);
end;

procedure TDDGBugsDataModule.wbdpBugswaRetrieveBugAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
{ 버그 상세 정보를 표시하는 페이지를 만들어낸다. }
var
  BugID: Integer;
  WebPage: TStringList;

procedure GetBug;
begin
  if tblBugs.Locate('BugID', BugID, []) then
    with tblBugs do
    begin
      WebPage.Add(Format('버그 ID:        %d', [BugID]));
      WebPage.Add(Format('<BR>보고자:   %s',
              [FieldByName('UserNameLookup').AsString]));
      WebPage.Add(FormatDateTime('"<BR>보고 날짜:"   mmm dd, yyyy',
        FieldByName('WhenReported').AsDateTime));
      WebPage.Add(Format('<BR>처리자:   %s',
              [FieldByName('AssignedToLookup').AsString]));
      WebPage.Add(Format('<BR>상태:        %s',
              [FieldByName('StatusTitle').AsString]));
      WebPage.Add(Format('<BR>요약:       %s',
              [FieldByName('SummaryDescription').AsString]));
      WebPage.Add(Format('<BR>세부 사항:       %s',
              [FieldByName('Details').AsString]));
```

```
      WebPage.Add('<BR>');
      WebPage.Add('<BR>');

      GetActions(WebPage);
    end;
end;

begin
  BugID := StrToInt(Request.QueryFields.Values['BugID']);

  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    with WebPage do
    begin
      Add('<BODY>');
      Add('<H1>버그 세부 사항</H1>');
      GetBug;
      AddFooter(WebPage);
      Response.Content :=  WebPage.Text;
      Handled := True;
    end;
  finally
    WebPage.Free;
  end;

end;
```

새 버그 정보 추가

사용자 로그인을 통과하면 나오는 HTML 페이지에는 새로운 버그를 데이터베이스에 추가하는 페이지로 가는 링크가 있었다. 그럼 사용자가 입력한 버그 정보를 데이터베이스에 추가하고, 다시 사용자에게 추가된 버그 정보를 표시해 주는 기능을 어떻게 구현하는지 살펴보자.

사용자가 입력한 정보 얻기

목록 33.11에 나온 TDDGBugsDataModule.wbdpBugswaGetBugInfoAction() 이벤트 처리 메소드는 사용자가 새로운 버그 정보를 입력할 수 있는 페이지를 만들어낸다. 이 메소드가 만들어내는 페이지에는 버그 정보를 입력하는 데 필요한 여러 가지 컨트롤들로 구성된 HTML 양식(form)이 포함된다. 그림 33.7이 이 메소드가 만들어낸 HTML 페이지이다.

목록 33.11 TDDGBugsDataModule.wbdpBugswaGetBugInfoAction()

```
procedure TDDGBugsDataModule.wbdpBugswaGetBugInfoAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
{ 새 버그 정보를 입력하기 위한 페이지를 만들어낸다. }
var
  WebPage: TStringList;

procedure AddAssignToNames;
{ HTML 페이지에 Assign To: 드롭다운 목록을 추가하고 목록을 채운다. }
begin

  WebPage.Add('<BR>처리자:');
  WebPage.Add('<BR><SELECT name="AssignTo"><BR>');

  with tblUsers do
  begin
    First;
    while not Eof do
    begin
      WebPage.Add(Format('<OPTION>%s %s - %s',
              [FieldByName('UserFirstName').AsString,
```

```
        FieldByName('UserLastName').AsString,
              FieldByName('UserName').AsString]));
      tblUsers.Next;
    end;
    WebPage.Add('</SELECT>');
  end;
end;

procedure AddStatusTitles;
{ HTML 페이지 상태 드롭다운 목록을 추가하고 목록을 채운다. }
begin
  WebPage.Add('<BR>상태:');
  WebPage.Add('<BR><SELECT name="Status"><BR>');

  with tblStatus do
  begin
    First;
    while not Eof do
    begin
      WebPage.Add(Format('<OPTION>%s',
          [FieldByName('StatusTitle').AsString]));
      tblStatus.Next;
    end;
    WebPage.Add('</SELECT>');
  end;
end;

begin
  WebPage := TStringList.Create;
  try
    AddHeader(WebPage);
    with WebPage do
    begin
      Add('<BODY>');
      Add('<H1>새 버그 추가</H1>');
      Add('<FORM action="../DDGWebBugs.dll/AddBug" ' +
              'method="GET">');
      Add('<BR>요약 설명:<BR><INPUT type="text" ' +
```

```
            'name="Summary" maxlength="100" size="50">');
        Add('<BR>세부 사항:<BR><TEXTAREA name="Details" ' +
            'rows=5 cols=50> </TEXTAREA>');

        AddAssignToNames;
        AddStatusTitles;

        Add('<p><INPUT type="SUBMIT"><INPUT type="RESET"></p>');
        Add('</FORM>');
        AddFooter(WebPage);
        Response.Content :=  WebPage.Text;
        Handled := True;
      end;
    finally
      WebPage.Free;
    end;
  end;
```

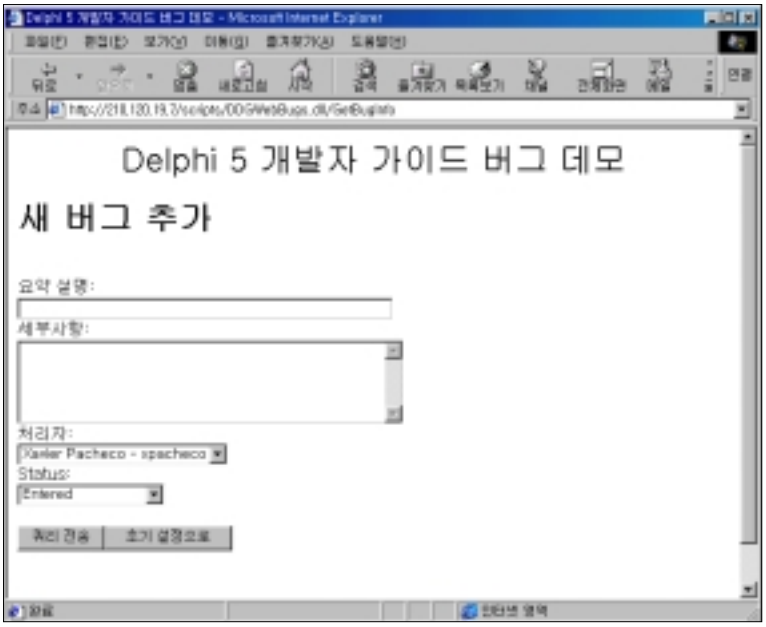


그림 33.7
버그 입력 페이지

AddAssignToNames()와 AddStatutsTitle()은 사용자의 선택을 편하게 하기 위한 두 개의 드롭다운 목록 상자를 만든다. 델파이 폼에서는 어떤 필드에 넣을 수 있는 값들을 제시할 때 데이터 인식 참조 컨트롤을 사용해서 자동적으로 목록을 채우지만, HTML 양식에서는 그것이 불가능하므로 직접 테이블의 레코드들을 처리해서 목록 상자에 채워야 한다. 그럼 사용자가 HTML 양식에 입력/선택한 값들을 읽어서 데이터베이스에 저장하는 메소드를 살펴보자.

버그 레코드 추가와 확인 페이지 표시

목록 33.12는 TDDGBugsDataModule.wbdpBugswaAddBugAction() 메소드이다. 버그 입력 페이지에서 Submit Query 버튼을 클릭하면 이 메소드가 수행된다.

목록 33.12 TDDGBugsDataModule.wbdpBugswaAddBugAction()

```
procedure TDDGBugsDataModule.wbdpBugswaAddBugAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
{ 버그 정보를 데이터베이스에 추가한다. 클라이언트가 돌려준 쿠키를 이용해서
  사용자에 대한 정보를 표시한다. }
var
  SummaryStr,
  DetailsStr,
  AssignToStr,
  StatusStr: String;
  WebPage: TStringList;
  UserID: Integer;
  UserName: String;
  UserFName,
  UserLName: String;
  AssignedToUserName: String;
  PostSucceeded: boolean;

function GetAssignedToID: Integer;
var
  PosIdx: Integer;
begin
  PosIdx := Pos('-', AssignToStr);
  AssignedToUserName := Copy(AssignToStr, PosIdx+2, 100);
  tblUsers.Locate('UserName', AssignedToUserName, []);
```

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

function GetStatusID: Integer;
begin
  tblStatus.Locate('StatusTitle', StatusStr, []);
  Result := tblStatus.FieldByName('StatusID').AsInteger;
end;

procedure DoPostSuccessPage;
begin
  with WebPage do
  begin
    Add(Format('<H1>감사합니다. %s %s, 당신의 버그가 추가되었습니다.</H1>',
          [UserFName, UserLName]));
    Add(FormatDateTime('"<BR><BR>버그 입력 날짜:"  mmm dd, yyyy', Date));
    Add(Format('<BR>버그 처리자: %s', [AssignedToUserName]));
    Add(Format('<BR>세부 사항: %s', [DetailsStr]));
    Add(Format('<BR>상태: %s', [StatusStr]));
  end;
end;

procedure DoPostFailPage;
begin
  WebPage.Add('<BR>버그 엔트리 실패.');
end;

begin

  // HTML 양식에 입력/선택된 값들을 얻는다.
  SummaryStr  := Request.QueryFields.Values['Summary'];
  DetailsStr  := Request.QueryFields.Values['Details'];
  AssignToStr := Request.QueryFields.Values['AssignTo'];
  StatusStr   := Request.QueryFields.Values['Status'];


  // 쿠키의 정보를 얻는다.
  UserID   := StrToInt(Request.CookieFields.Values['UserID']);
  UserName := Request.CookieFields.Values['UserName'];
  UserFName := Request.CookieFields.Values['UserFirstName'];
```

```
UserLName := Request.CookieFields.Values['UserLastName'];

// AfterInsert 이벤트 처리 메소드에서 사용할 값들을 배정
FLoginUserID   := UserID;
FLoginUserName := UserName;

InsertBug;
try
  tblBugs.FieldByName('SummaryDescription').AsString := SummaryStr;
  tblBugs.FieldByName('WhenReported').AsDateTime := Date;
  tblBugs.FieldByName('Details').AsString := DetailsStr;
  tblBugs.FieldByName('AssignedToUserID').AsInteger := GetAssignedToID;
  tblBugs.FieldByName('StatusID').AsInteger := GetStatusID;
  tblBugs.Post;
  PostSucceeded := True;

except
  tblBugs.Cancel;
  PostSucceeded := False;
end;

WebPage := TStringList.Create;
try
  AddHeader(WebPage);
  with WebPage do
  begin
    Add('<BODY>');

    if PostSucceeded then
      DoPostSuccessPage
    else
      DoPostFailPage;

    AddFooter(WebPage);
    Response.Content :=  WebPage.Text;
    Handled := True;
  end;
finally
  WebPage.Free;
end;
```

```
end;

end.
```

이 이벤트 처리 메소드는 그림 33.7에 나온 버그 입력 페이지에 사용자가 입력한 값들을 얻는다. 그리고 이전에 쿠키에 기록했던 값들도 얻는다. 다음 두 줄의 코드는 tblBugs의 AfterInsert 이벤트 처리 메소드 때문에 필요한 것이다.

```
// AfterInsert 이벤트 처리 메소드에서 사용할 값들을 배정
FLoginUserID   := UserID;
FLoginUserName := UserName;
```

tblBugs의 AfterInsert 이벤트 처리 메소드에서는 다음과 같은 작업을 한다.

```
tblBugs.FieldByName('UserID').AsInteger := FLoginUserID;
tblBugs.FieldByName('UserNameLookup').AsString := FLoginUserName;
```

필요한 값들을 얻은 후에는 tblBugs에 새 버그 레코드를 추가한다. 만일 레코드 추가가 성공하면 DoPostSuccessPage()를 호출하고, 성공하지 못하면 DoPostFailPage()를 호출한다. DoPostSuccessPage()는 사용자가 입력했던 값들을 다시 표시해 주는 페이지를 만들어내며, DoPostFailPage()는 작업이 실패했다는 페이지를 만들어낸다.

버그 입력 폼의 두 목록 상자의 내용은 데이터 인식 컨트롤로 채운 것이 아니라는 점을 기억할 것이다. 데이터 인식 컨트롤이 사용되지 않았기 때문에 tblBugs에 추가할 적절한 참조 색인값을 얻으려면 tblUsers와 tblStatus 테이블에서 직접 적절한 값을 찾아내야 한다. 이 메소드는 사용자가 두 드롭다운 목록 상자에서 선택한 문자열을 이용해서 검색을 수행하는데, HTML 양식의 두 목록 상자에 있는 문자열들은 실제 테이블에 있는 문자열들과 약간 다른 형태이므로 적절한 문자열 처리 작업을 통해서 정확한 문자열을 만들어낸다(GetAssignToID() 메소드를 보기 바란다).

요약

이번 장에서는 웹 데이터베이스 애플리케이션을 만들어 보았다. 또한, 기존의 애플리케이션(물론, 웹을 염두에 두고 설계한 것이다)에 웹 관련 코드만을 추가해서 웹 기반의 애플리케이션으로 변신시키는 방법을 제시했다. 이번 장에서 추가된 코드들은 대부분 데이터베이스를 조작하는 것들이 아니라 HTML 페이지를 다루는 것들이었다. 독자 나름대로 데이터 모듈을 좀더 개선하거나 코드의 HTML 생성 부분을 실제 HTML 파일로 대치시켜 보는 것도 재미있을 것이다.

쉬어가는 페이지