

Effective Data Handling In Web Applications

by Steve Scott

This paper examines a number of issues involved in developing web-based applications that require data from a database of some sort. It assumes that you have some experience of web development and you have a working knowledge of the Delphi WebBroker/WebSnap Technology. Note: The examples I use here often use hard coded paths for simplicity; it is obviously not recommended that you do this in real applications.

Why is Effective Data Handling Important?

First we must ask ourselves why it is so essential to have effective data handling in our web application. As traditional application developers we have become use to working on reasonably fast 10MB, 100MB or maybe even GIGABYTE based networks. When the user presses a button on one of our applications' forms they may have to wait one or even two seconds to see the result. Often the majority of this time is taken up with database activity.

If you now move our application to the web, this introduces a whole new time delay in the form of the often limited bandwidth between the user's browser and the web server. This bandwidth limitation can often introduce a three, four or even five second delay into any transaction. Now add the normal two seconds of data processing and we are starting to talk about a time period that users will not find acceptable.

Added to this, a normal Windows application can maintain a query result for as long as it exists whereas, in the stateless world of web development, a query has to be run over and over again often just to get the next few records. Because the very nature of stateless development makes databases inefficient we must do all we can not to make the situation any worse.

Don't Do It Unless You Have To

Before looking at different database issues and data handling techniques, I cannot stress strongly enough that your first approach to your web application should be to avoid data access wherever possible. Data access is processor heavy, complex and one of the biggest reasons web applications often don't scale very well.

Get rid of the database

Imagine an application that has to display a simple form that contains a drop down box. The values for this box are kept in a database. If you just dive right in and write your application to populate this box from your database using page producers and custom tags, you are performing a database access every time you display that form. If you are writing a CGI application you will also have to make a database connection each time the form is displayed. It is often the case that values stored in databases to act as lookup data are fairly static and only change occasionally. This means our application could be making a database connection and performing queries to examine data that might change only once a day or even less. If you are writing ISAPI based applications you may consider caching the results between calls but with CGI you don't have this option.

In cases like this it is best to avoid using a database within the web application altogether. Consider using a separate program (probably a service) that monitors infrequently changing data in a database. When data changes, this program can totally re-create any template html files the application uses.

In our web application we continue to use custom tags but, rather than perform a database lookup, we simply replace the tag with a file from the disk using another page producer as a mechanism of reading this file.

Using this solution, the previously described web application can now show its form without a single database connection or access being required. A lot faster, simpler and more robust.

Example

```
procedure
TemplateDemo.PageProducerHTMLTag(Sender:
TObject; Tag: TTag; const TagString: String;
TagParams: TStrings; var ReplaceText: String);
begin
    if tagstring = 'DROPDOWNOPTIONS' then begin
        pgTemplates.HTMLFile :=
            'c:\dropdowntemplate.htm';
        replacetext := pgTemplates.Content;
    end;
end;
```

See wpgTemplateDemo.Pas for full listing

Another example of using this technique is to produce complete HTML pages in advance rather than produce them on the fly from a database. For example, take a database full of real estate brochures. Why do you need to generate the brochure every time someone requests it, when it probably doesn't change very often? Again, using an external program, you can monitor the data (maybe not even on the web server) and create new HTML pages for any real estate brochure that changes. In the web application simply use the HTMLFile property of a page producer to load the correct file.

Example

```
unit wpgRealEstateBrochure;
interface
uses
  Windows, Messages, SysUtils, Classes,
  HTTPApp, WebModu, HTTPProd;
type
  TRealEstateBrochure = class(TWebPageModule)
    PageProducer: TPageProducer;
    procedure WebPageModuleActivate(Sender:
      TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

  function RealEstateBrochure:
    TRealEstateBrochure;
implementation
  {$R *.dfm}

uses WebReq, WebCntxt, WebFact, Variants;

function RealEstateBrochure:
  TRealEstateBrochure;
begin
  Result :=
  TRealEstateBrochure(WebContext.FindModuleClass
    (TRealEstateBrochure));
end;

procedure TRealEstateBrochure.WebPageModule
  Activate(Sender: TObject);
var
  BrochureNo: String;
begin
  case Request.MethodType of
    mtGet: BrochureNo := Request.QueryFields.
      Values['brochureno'];
    mtPost: BrochureNo := Request.QueryFields.
      Values['brochureno'];
  end;
  PageProducer.HTMLFile := 'c:\brochure' +
    BrochureNo + '.htm';
end;

initialization
  if WebRequestHandler <> nil then
    WebRequestHandler.AddWebModuleFactory(
      TWebPageModuleFactory.Create(
        TRealEstateBrochure,
        TWebPageInfo.Create([wpPublished
          {, wpLoginRequired}], ''),
        crOnDemand, caCache));
end.
```

See `wpgRealEstateBrochure.Pas`

Use the database only when you have to

Sometimes reproducing whole html pages is just impractical. For example, if you have a site that allows people to perform some kind of searching, it is not possible to prepare previously created pages of search results. Obviously in this case we have to use a database of some sort in order to produce a list of required results.

Often even in this scenario, even though the required data (i.e. the search results) may not be constant, the actual individual results items may be. For example, on the web site that allows you to search for real estate brochures based on a number of criteria, the actual list of matching real estate brochures will always be different but the real estate brochures themselves are fairly static.

In this case we can use a combined technique where all a database search needs to return is a list of the real estate

brochure keys required. The individual real estate brochures are then read in from previously created HTML files that will be recreated only when that real estate brochure actually changes.

So, to conclude this section, your first approach to effective data handling in your web application is to remove all unnecessary database access.

Methods and Techniques

Stand Alone TClientDataSet

If you have data that needs to be searched but not updated and the search data rarely changes (my definition of rarely is a timescale of hours rather than minutes) then using TClientDataSet as an in-memory table can be very efficient. The table will be initially loaded from the file when the Web Module/Web Data Module on which it is located is created. The client dataset can be optimised to its required usage by indexes relevant to the required task being added on the fly. The required result set can be obtained using filters.

Obviously using this method requires a mechanism to be in place to reload the disk based table if it has changed. How the table is altered on the disk is an entirely separate issue, but one possible technique is that the table is recreated by a service in a similar way to the HTML Templates discussed earlier. In your Web Module/Web Data Module you could periodically check if the Date & Time has changed on the disk based table and, if it has, reload.

Example

```
unit uwdmDataHandlingClientDataSet;
interface
uses
  Windows, Messages, SysUtils, Classes,
  HTTPApp, WebModu, DB, DBClient;
type
  TwdmDataHandlingClientDataSet =
    class(TWebDataModule)
      cdsData: TClientDataSet;
      procedure WebDataModuleCreate(Sender:
        TObject);

  private
    { Private declarations }
    FLastFileCheckTime: Real;
    FLastLoadedTime : Integer;
    procedure LoadDataFile;
    procedure CheckFile;
  public
    { Public declarations }
    function CustomersLike(strPattern:
      string): string;
  end;

  function wdmDataHandlingClientDataSet:
    TwdmDataHandlingClientDataSet;
implementation
  {$R *.dfm}

uses WebReq, WebCntxt, WebFact, Variants;
const
  FILECHECKTIME = 0.041666; // About 1 Hour
function wdmDataHandlingClientDataSet:
  TwdmDataHandlingClientDataSet;
begin
  Result :=
  TwdmDataHandlingClientDataSet(WebContext.
    FindModuleClass(TwdmDataHandlingClient
      DataSet));
end;
```

```

procedure
TwdmDataHandlingClientDataSet.CheckFile;
var
  sr: TSearchRec;
begin
  if Now - FLastFileCheckTime > FILECHECKTIME
  then begin
    if FindFirst('C:\BorconData2002.dat', 0,
      sr) = 0 then begin
      if sr.Time <> FLastLoadedTime then
        LoadDataFile;
      FindClose(sr);
    end;
  end;
  FLastFileCheckTime := Now;
end;

function
TwdmDataHandlingClientDataSet.CustomersLike
(strPattern: string): string;
begin
  CheckFile;
  cdsData.Filter := 'CustName="' + strPattern
    + '""';
  cdsData.Filtered := True;
  cdsData.First;
  while not cdsData.eof do begin
    Result := Result + cdsData.FieldByName
      ('CustName').AsString + ' ';
    cdsData.Next;
  end;
end;

procedure
TwdmDataHandlingClientDataSet.LoadDataFile;
begin
  cdsData.Close;
  cdsData.LoadFromFile('C:\BorconData2002.
  dat');
  cdsData.Open;
  cdsData.AddIndex('idxCustNo', 'CustNo',
    [ixUnique])
end;

procedure
TwdmDataHandlingClientDataSet.WebDataModuleCreate
(Sender: TObject);
begin
  FLastFileCheckTime := 0;
  FLastLoadedTime := 0;
end;

initialization
  if WebRequestHandler <> nil then
    WebRequestHandler.AddWebModuleFactory(
      TWebDataModuleFactory.Create
        (TwdmDataHandlingClientDataSet,
          crOnDemand, caCache));
end.

```

The general usefulness of this technique will depend on the type of data being searched. The data obviously needs to be small enough to hold in memory. If you are using data for searching purposes try to use integers and short fields as much as possible: this will allow even tables with a large number of records to be loaded into memory quite quickly.

Queries

Query Design

Query results are handled differently in web applications than in a normal Client/Server application. It may be acceptable to run a query from a desktop app that will return thousands of records, especially if it is going to remain open for a long while. In a web application, that query result will have a life of just a few seconds and will probably be run over and over again. You should design your queries always to return the smallest result set possible in the fastest time.

Let's take a look at an application that supplies a list of orders from the orders table. Each page will display ten orders and offer a next and back button for the next and previous pages of orders. In a Client/Server application, you would write a query that returns all the orders, display the results in a DBGrid or similar control, and let the control fetch the data as and when it was required. No matter how many times the user scrolled up and down the data, the query has only been run once. In a web application the query will have to be run every time a new page is required, regardless of whether that page has been displayed before or not. Once the query has run, the web application will need to scroll to the right place to get the next or previous ten records. Anything you can do to reduce the time the query takes to run or reduces the number of records returned could dramatically improve performance. (This is even the case using WebSnap, the DataSet Adapter performs this operation for you.)

Queries should never return more fields than they actually require. If your database has any mechanisms built in that allow you to restrict the number of rows returned, then you should also take advantage of this.

Uni-Directional Result Sets

If your data access technology supports uni-directional result sets then you should take advantage of this as much as possible. DBExpress queries are only uni-directional which makes DBExpress an excellent choice for Web Development.

Design

Database Design

The next biggest area that will allow more effective data handling in your web application is the actual design of your database. Web applications will often use a database in a totally different way to your traditional Client/Server or Desktop application. Design your database carefully. What may be good in a traditional application may actually be a performance hindrance in a web application.

Proper Indexing is vital because speed is of the essence when you already have the overhead of the Internet to deal with. If designing specifically for the Web, then consider using extra fields just for index and ordering purposes.

Normalization was invented for data integrity reasons not performance reasons. If designing only for the web, then you need to consider the performance advantages you gain through de-normalization. This will depend on how much data writing your web application has to deal with.

Data Access Technology

The choice of middleware you use to access the data from your web application can greatly affect your application's performance.

BDE

The BDE is not good for Web Applications. Designed as a flexible client-side piece of middleware, it does not work well in a heavily multi-threaded server-side environment. It has serious scalability problems and its size make it inefficient, especially for CGI work. The BDE is an excellent piece of software in many situations but Web development is not one them.

ADO

ADO, like the BDE, is actually a fairly large inefficient piece of middleware and its suitability for Web Development will depend very heavily on the database and OLE DB driver you use with it. In many cases, when using ODBC or MS Access, it is no better than the BDE. Where it does win over the BDE is with its connection pooling. If you use ADO against SQL Server then you will be able to take advantage of the fact that connections with the same parameters will be pooled and shared. This can bring the amount of time it takes to establish a connection to virtually nil.

DBExpress

DBExpress can make an excellent choice of middleware for Web Development. As a very light wrapper around a native driver, DBExpress itself adds virtually no overhead and, with its default uni-directional result sets, aids efficiency. Obviously, how effective it is will depend on the efficiency of the underlying database, but DBExpress will bring as close to using that driver as any middle solution is likely to be able to achieve.

Other Databases/Tools

If you are using third party database middleware or engines, then there are a number of key areas you need to examine. The first is how well the technology operates in a multi-threaded environment. If the database technology can handle many simultaneous accesses efficiently, then it is likely to work well in a Web Application environment. Other factors include how long it takes to establish a connection. File based databases should only be used if they can be hosted on the same machine as the database, as they traditionally operate very inefficiently in a network environment.

Don't forget the use to which you put your database may dramatically affect its performance. Some databases will work very well in a read-only environment but quickly slow down as soon as any updating takes place.

Application Formats

CGI

Using a database in a CGI application is something that should be avoided whenever possible. As the program has to start every time a request is run, database connections have to be created and established for every request. If you really must use a database in a CGI application, then try and use a small lightweight database that will compile directly into your executable so that no other DLLs or Packages have to be loaded or initialised each time the program is run. Client/Server databases should be avoided, as they require a logon and password process to be undertaken each time a connection is made.

CGI applications are not ideal when it comes to data handling, yet often CGI is the developer's only option, as ISAPI is not allowed by many ISPs.

ISAPI/NSAPI/Windows Apache DSO

Because, when writing Web Broker/WebSnap based ISAPI applications our Web Modules/Web Data Modules are created and then cached, we have better options when dealing with databases.

Don't forget that these applications are multi-threaded. Web Modules and Web Data Modules are created on a

thread, run and then put in a global cache. The next request that comes in will be allocated a thread that will take a Web Module/Web Data Module from the global cache to operate on. This means our Web Module objects during their lifetime will run on many different threads. Because of this thread jumping behaviour, global database connections are best not used. Each Web Module/Web Data Module instance should be totally self-contained with its own database instance. If using the BDE, each Web Module/Web Data Module should also have its own session component with the AutoSessionName property set to true. It is best if connections are created and opened when the Web Module/Web Data Module is first created and closed, and destroyed when the Web Module/Web Data Module is destroyed. You should always include error-handling code in your application that will deal with the fact that a cached connection may have broken.

Generic Database Issues

Location

Databases and Web Servers should not reside on the same machine for a number of reasons. Firstly, both applications like to hog the machine's memory and have good access to the machine's hard disks. Running both on the same machine can dramatically reduce the performance of both. Another good reason not to keep your database on the web server is security. Web Servers by their very nature tend to be exposed machines. Placing your data on the same machine exposes your data to greater risk of unauthorised access. Databases should be on another machine, safely tucked behind a firewall. Unfortunately, this very act will also reduce performance, as query results will now need to be transmitted across a network rather than handled in local memory.

Database Load

Uncontrollable connections are another performance issue when dealing with database enabled web applications. In a Client/Server world you can control the number of connections made to any given database. In a web world, where Web Module Instances containing connection components are being created, you have far less control over the number of connections being made to the database. This can cause a licensing issue but, more importantly, may have a performance issue. There is probably an optimum number of connections at which your particular database might run. As you go over this number, performance can degrade quite dramatically. When you have no control over the number of connections being made it is very hard to get the database to work at its most efficient. For example, it can often be far quicker to queue a hundred query requests and run twenty five at a time than to try and run all hundred at once.

Summary

As you have no doubt realized it is only worth trying to optimize the actual code you write when dealing with a database application in a web environment if you have first considered a large number of issues. Your choice of database, application architecture and data & query design can only work efficiently when considered all together rather than in isolation.

