Appeon Performance Tuning Guide

Appeon® 6.0 for PowerBuilder®

DOCUMENT ID: DC10089-01-0600-03

LAST REVISED: August 2008

Copyright © 2008 by Appeon Corporation. All rights reserved.

This publication pertains to Appeon software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Appeon Corporation.

Appeon, the Appeon logo, Appeon Developer, Appeon Enterprise Manager, AEM, Appeon Server and Appeon Server Web Component are trademarks or registered trademarks of Appeon Corporation.

Sybase, Adaptive Server Anywhere, and PowerBuilder are trademarks or registered trademarks of Sybase, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Appeon Corporation, 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

1 About This Book 1
1.1 Audience 1
1.2 How to use this book 1
1.3 Related documents 1
1.4 If you need help 2

2 Appeon 6.0 Performance 4
2.1 Expected performance level 4
2.2 Automatic performance boosting 4
2.3 Impact of the Internet and slow networks 5
2.4 Impact of heavy client-side logic 6
2.5 Impact of large data transmission 7

3 Identifying Performance Bottlenecks 8
3.1 Overview 8
3.2 Heavy window report 8

4 Performance-Related Settings 9
4.1 Overview 9
4.2 Appeon Developer performance settings 9
4.3 Appeon Enterprise Manager performance settings 9
4.3.1 DataWindow data caching 9
4.3.2 Multi-thread download settings 9
4.3.3 Custom libraries download settings 10
4.3.4 Log file settings 10
4.4 Internet Explorer performance settings 10
4.5 Web and application server performance settings 10
4.5.1 Sybase EAServer 10

5 Tuning: Excessive Server Calls 12
5.1 Overview 12
5.2 Technique #1: partitioning transactions via stored procedures 12
5.3 Technique #2: partitioning non-visual logic via NVOs 14
5.4 Technique #3: eliminating recursive Embedded SQL 16
5.5 Technique #4: grouping multiple server calls with Appeon Labels 18

6 Tuning: Heavy Client 22
6.1 Overview 22
6.2 Technique #1: thin-out “heavy” Windows 22
6.3 Technique #2: thin-out “heavy” UI logic 22
6.3.1 Manipulating the UI in loops 22
6.3.2 Triggering events repeatedly 23
6.3.3 Performing single repetitive tasks 23
6.3.4 Initializing “heavy” tabs 24
6.3.5 Using ShareData or RowsCopy/RowsMove for data synchronization 24
6.3.6 Using computed fields 24

6.3.7 Using DataWindow expressions 25
6.3.8 Using complex filters 25
6.3.9 Using RowsFocusChanging/RowsFocusChanged events 25
6.4 Technique #3: offload “heavy” non-visual logic 25

7 Tuning: Large Data Transmissions 26
7.1 Overview 26
7.2 Technique #1: retrieving data incrementally 26
7.2.1 For Oracle database server 26
7.2.2 For all other database servers 27
7.3 Technique #2: minimizing excessive number of columns 27

8 Conclusion 28

Index 29

1 About This Book

1.1 Audience

This book is intended to help developers plan what steps they will take, and how much time they will invest in improving the performance of a PowerBuilder application deployed to the Web with Appeon 6.0. It is also intended to guide PowerBuilder architects and developers on how to build new PowerBuilder applications that perform well when deployed to the Web or a WAN.

1.2 How to use this book

There are five chapters in this book:

Chapter 1: About This Book

A general description of the contents of this document.

Chapter 2: Appeon 6.0 Performance

Describes current runtime performance levels of Appeon 6.0 and primary reasons for performance issues (if any).

Chapter 3: Identifying Performance Bottlenecks

Describes approach for identifying areas in the Web application that may suffer from runtime performance issues. Also, describes the Appeon Developer performance reporting tool - Heavy Window Report.

Chapter 4: Performance-Related Settings

Documents key Appeon, Web browser, and application server settings that should be configured from default values to ensure optimal performance of your Web system.

Chapter 5: Tuning: Excessive Server Calls

Introduces the performance tuning concept "Excessive Server Calls" with several techniques to optimize your PowerBuilder code to achieve good Web performance.

Chapter 6: Tuning: Heavy Client

Introduces the performance tuning concept "Heavy Client" with several techniques to optimize your PowerBuilder code to achieve good Web performance.

Chapter 7: Tuning: Large Data Transmissions

Introduces the performance tuning concept "Large Data Transmissions" with several techniques to optimize your PowerBuilder code to achieve good Web performance.

Chapter 8: Conclusion

Final thoughts and recommendations on performance tuning of your PowerBuilder applications for the Web.

1.3 Related documents

Appeon provides the following user documents to assist you in understanding Appeon for PowerBuilder and its capabilities:

- *Appeon Demo Applications Tutorial*:

Introduces Appeon's demo applications, including the Appeon Sales Application Demo, Appeon Code Examples, Appeon ACF Demo, and Appeon Pet World, which show Appeon's capability in converting PowerBuilder applications to the Web.

- *Appeon Developer User Guide* (or *Working with Appeon Developer Toolbar*)

Provides instructions on how to use the Appeon Developer toolbar in Appeon 6.0.

Working with Appeon Developer Toolbar is an HTML version of the *Appeon Developer User Guide*.

- *Appeon Server Configuration Guide*

Provides instructions on how to configure Appeon Server Status Monitor, establish connections between Appeon Server and Database Server, and configure AEM for maintaining Appeon Server and Appeon deployed Web applications.

- *Appeon Supported Features Guide* (or *Appeon Features Help*):

Provides a detailed list of what PowerBuilder features are supported and can be converted to the Web with Appeon 6.0 and what features are unsupported.

Appeon Features Help is an HTML version of the *Appeon Supported Features Guide*.

- *Appeon Installation Guide*:

Provides instructions on how to install *Appeon for PowerBuilder* successfully.

- *Appeon Migration Guide*:

A process-oriented guide that illustrates the complete diagram of the Appeon Web migration procedure and various topics related to steps in the procedure, and includes a tutorial that walks the user through the entire process of deploying a small PowerBuilder application to the Web.

- *Appeon Performance Tuning Guide*:

Provides instructions on how to modify a PowerBuilder application to achieve better performance with its *corresponding Web application*.

- *Appeon Troubleshooting Guide*:

Provides information about troubleshooting issues, covering topics such as product installation, Web deployment, AEM, Web application runtime, etc.

- *Introduction to Appeon*:

Guides you through all the documents included in Appeon 6.0 for PowerBuilder.

- *New Features Guide* (or *What's New in Appeon*):

Introduces new features and changes in Appeon 6.0 for PowerBuilder.

What's New in Appeon is an HTML version of the *New Features Guide*.

1.4 If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support or an Authorized Sybase Support Partner. If you have any questions about this product or if you need assistance

during the installation process, ask the designated person to contact Sybase Technical Support or an Authorized Sybase Support Partner based on your support contract. You may access the Technical Support Web site at http://www.sybase.com/support.

2 Appeon 6.0 Performance

2.1 Expected performance level

When comparing a PowerBuilder application to the performance of a Web application deployed by Appeon to a LAN environment, generally speaking the performance of the two will be quite similar. In some cases (for certain operations) Appeon may actually be even faster than PowerBuilder.

The reason is that Appeon has been tuned for nearly a decade to offer the best performance possible for real-life PowerBuilder applications:

- Large PowerBuilder applications up to 600MB (of PBLs) including several thousand DataWindows and thousands of Windows.
- Complex screens containing as much as 80 DataWindows in a single Window
- Complex tabs (including nested tabs and dynamically created tabs)
- Dynamically created objects (DataWindows, UserObjects, etc.)
- PFC and other high-overhead frameworks similar to PFC

2.2 Automatic performance boosting

Appeon has a number of features built into its infrastructure/framework to instantly or automatically boost the performance of PowerBuilder applications when deployed to the Web. Many of these features are always on and transparently working in the background to boost performance. Other features are user-selectable and must be configured. Table 2-1 is list of these features and the configuration of these features is covered in *Chapter 4: Performance-Related Settings.*

Table 2-1: Appeon Performance Boosting Features

Performance Feature	Description	Location
Just-in-Time Downloading	As the application is run and various windows are opened, only the Web files required for that particular window are downloaded at that point in time. Once the Web files are downloaded, they are cached in the Internet Explorer temporary files folder and are not downloaded again.	Appeon Infrastructure
10X Web File Compression	All JavaScript files are compressed by as much as 10X, then the compressed version of the file is downloaded over HTTP to the Web browser.	Appeon Developer
10X Data File Compression	For each DataWindow or DataStore retrieval, the result set is first retrieved by the application server, automatically compressed by 10 times in most cases, and then downloaded over HTTP to the Web browser. Utilizing AJAX technology, only the DataWindow or DataStore is refreshed and the rest of the screen remains intact.	Appeon Infrastructure

DataWindow Data Caching	For each DataWindow or DataStore, the developer has the option of enabling caching of the result set. Appeon enables caching at the application server, Web server, and Web browser so every tier of the Web architecture is benefiting from the best performance and scalability possible.	Appeon Enterprise Manager
Merge files	Merges multiple JavaScript files into a single file to reduce the number of HTTP requests and corresponding overhead.	Appeon Developer
Multi-thread Downloading	Downloads are multi-threaded to boost the application runtime performance.	Appeon Enterprise Manager
Custom Libraries Downloading	Any custom libraries can be automatically downloaded and installed with your Web application, or if the libraries are very large in size, you can disable this feature and distribute the libraries some other fashion.	Appeon Enterprise Manager
Database Connection Pooling	By deploying to a true n-tier Web environment with Appeon, you can take advantage of Database Connection Pooling, a feature of most application servers. Connection Pooling does exactly that, it establishes a pool of connections to your database, which is shared among your clients. So rather than each client having its own dedicated connection to the database, a fewer number of connections can be rotated among all the users. For large deployments with thousands of clients this can boost database scalability noticeably.	Appeon Infrastructure

2.3 Impact of the Internet and slow networks

Although Appeon pushes the envelope to deliver unparalleled performance from standard Web technologies (e.g. XML, JavaScript, HTML, Java or C#), which are typically significantly slower than PowerBuilder, slow and latent network connections rob performance from even the best applications!

Network chatter and network-intensive code really highlight the weakness of a poor network connection. Any code that results in a HTTP request (i.e. server call) when executed multiple times sequentially has potential to create network chatter. There are mainly two categories of code that result in server calls - data access related and remote method invocations. Here are several common examples so you can familiarize yourself:

- Embedded SQL (Select, Insert, Delete, Update, Cursor) including Dynamic SQL;
- Invoking stored procedures or database functions;
- DataWindow/DataStore Functions (Retrieve, Update, ReselectRow, GetFullState, SetFullState, GetChanges, SetChanges);
- DataWindow/DataStore Events (SQLPreview);
- Invoking a method of a server-side object, such as a PowerBuilder NVO, Java EJB, or .NET Component; or
- Invoking a Web Service.

Each of the above statements will generate one call to the server utilizing HTTP, with exception of SQLPreview event that will generate one call for each line of code handled by the event. If any of the above statements are contained in a loop or recursive function, well

depending on the number of loops, even though its just one statement it would be executed multiple times generating multiple server calls. Needless to say, loops and recursive functions are some of the most dangerous from a performance perspective.

The reason it is important to minimize server calls is because it can take 100 or even 1,000 times longer to transmit one packet of data over the Internet compared to a LAN. Imagine an event handler is triggered, for example handling an "onClick" event, whose execution will result in 80 synchronous server calls over a LAN with latency of 2 milliseconds (ms). In such scenario the slow-down attributed to network latency would be 0.16 seconds (80 x 2 ms). Now imagine this same event handler running over a WAN with latency of 300 ms. The slow-down attributed to the network latency would be a whopping 24 seconds (80 x 300 ms)! And depending on the amount of data transmitted there could be additional slow-down due the bandwidth bottlenecks.

It is imperative for the developer to be conscious that PowerBuilder applications deployed to the Web may not be running in a LAN environment, and as such there will be some degree of performance degradation. How much depends on how the code is written, but in most cases the performance degradation still falls within acceptable limits without much performance optimization.

Should you find that certain operations in your application are unacceptably slow, the good news is there are numerous things that you can do as PowerBuilder developers to ensure your PowerBuilder applications perform well in a WAN environment (e.g. Internet) or on slower networks. At a high-level, your code needs to be written such that the server calls and other performance intensive code is minimized or relocated to the middle-tier or back-end. This will be covered in more detail in the following chapters. Some changes are actually quite simple while others may require increased effort. Nonetheless, in all cases optimizing the performance of your applications in PowerBuilder is just a fraction of the effort to work with typical low-productivity Web tools such as VisualStudio.NET and Eclipse.

2.4 Impact of heavy client-side logic

Most PowerBuilder applications are developed utilizing a 2-tier architecture. In other words, all the PowerScript and embedded SQLs are coded in the Visual objects, for example Window, CommandButton, etc. In contrast, a 3-tier architecture would encapsulate all non-visual logic in PowerBuilder NVOs (Non-Visual Objects). The reality is even if your application utilizes NVOs, chances are it is not a pure 3-tier application if PowerBuilder NVOs are not exclusively utilized to encapsulate all non-visual logic. But don't rush to partition your application just yet!

Most applications developed as a 2-tier architecture perform great in Appeon. In fact, there are many situations that a 2-tier application when deployed by Appeon will actually perform faster than a 3-tier application. The reason is if a PowerBuilder NVO is deployed to the middle-tier or application server, time must be spent to call the server and get the results back to the client. Of course, your non-visual logic running on an application server will run faster than at the Web browser. The key question is how much performance do you gain by running a particular block of code on the application server vs. how much performance do you lose due to the server calls.

As a rule of thumb, it is recommended to partition your non-visual logic to the middle-tier only when the particular block of code runs unacceptably slow at the Web browser. In such cases, it is likely that the application performance will benefit, and as such, it is worthwhile to invest the time to partition such logic. However, if the non-visual logic is only slightly

sluggish, it may be possible to optimize the code without having to partition it to the application server.

2.5 Impact of large data transmission

When you first open a Window there are two types of files downloaded. The first type is the HTML and JavaScript files ("Web files") that contain the UI and UI logic of the application Window. The second type is data files that contain the result set, for example for a DataWindow retrieve. The time to download these files is affected by two factors: 1) the network connection and 2) the size of the files to be downloaded.

The Web files do not impact performance because of their small size and the enhanced ability of the browser to "cache" them. The Web files for a given PowerBuilder Window are typically between 25-75 KB. Because these Web files are static in nature, once a given application Window has been opened, the Web files will be cached on the Client computer. As such, once these Web files are cached, their impact on performance is essentially non-existent.

Under most circumstances, these Web files are not re-downloaded when the Window is reopened. The only exceptions are if 1) the temporary Internet files folder has been emptied or 2) the application has been updated and redeployed to the server. If the latter has happened, Web files for only those Windows that have been modified will be automatically downloaded from the Web server.

Only the data files containing the result sets may or may not be cached (depending on whether you have enabled DataWindow caching). A result set of 50 records would typically result in a 12 KB data file. Every 5 records would typically add another 1.2 KB to the data file size. So, for example, a 500-record result set would typically correspond to a 120 KB data file. If DataWindow caching is not enabled, the data files corresponding with such DataWindows will be downloaded from the server each time a DataWindow retrieve is invoked.

The good news is that Appeon has built-in 10X data compression for DataWindow result set to essentially eliminate the time spent downloading these data files. The same 500-record result set that would normally correspond to a 120 KB data file would only result in the download of a 12 KB data file from the server. This compression feature makes even the largest of result sets quick to transfer.

In conclusion, due to Web file caching feature of the Web browser, Appeon's built-in DataWindow caching technology and 10X data compression technology, generally speaking neither the Web files nor data files should have any noticeable impact on the performance of your Web application.

3 Identifying Performance Bottlenecks

3.1 Overview

There are several methods to identify performance bottlenecks in your application. You can manually test your application or utilize Appeon's built-in performance reporting tool. Manually testing is the most time-consuming but also the most comprehensive and accurate. Nonetheless, the performance reporting tool, Heavy Window Report, can help to identify most problematic Windows without a lot of work.

3.2 Heavy window report

The Heavy Window Report is a tool in Appeon Developer to help identify "Heavy" Windows. Heavy Windows are Windows in the PowerBuilder application, which are likely to affect the performance of the deployed Web application. These Windows tend to be stuffed with data-intensive objects and code, such as DataWindows and Embedded SQL.

This is essentially a "yellow flag" tool to identify Windows that may exhibit performance issue. Generally speaking, Heavy Windows will only exhibit performance issues on slow and high-latency networks. Once a Heavy Window is identified, it is recommended to test such Window over the Web to confirm whether in fact the Window exhibits performance issues. When performing this Web testing, it is important to mimic your users' environment (e.g. network connection, hardware, etc.) as much as possible, to get an accurate reflection of what your users will ultimately experience.

The Heavy Window report is fully configurable so that you can adjust it to the demands of your network connection. You can specify how "heavy" a Window can be and what PowerBuilder features make the Window "heavy". Refer to the *Appeon Developer User Guide* for more information.

4 Performance-Related Settings

4.1 Overview

Performance settings need to be configured in Appeon Developer, Appeon Enterprise Manager, Web browser and your application server to ensure good performance in a production environment. If you identify any performance bottlenecks, it is strongly recommended that you first ensure all performance-related settings are correctly configured. Only if the performance issues still persist after the performance settings are configured correctly, then it is recommended to consider optimizing your PowerBuilder code.

4.2 Appeon Developer performance settings

Table 4-1 lists the Appeon performance settings that the developer can configure (in the application profile) of Appeon Developer to boost performance. To make a performance setting effective for an application, enable the option in the application profile and then perform a “Full Deployment” of the application.

Table 4-1: Performance settings in Appeon Developer

Performance Feature	Description	User-Selectable
10X Web File Compression	Compresses Web files when they are transferred over the network.	User must enable this feature for it to become effective.
Merge files	Merges multiple Web files into a single Web file to reduce the number of HTTP requests and corresponding overhead.	User must enable this feature for it to become effective.

4.3 Appeon Enterprise Manager performance settings

4.3.1 DataWindow data caching

Appeon Enterprise Manager (AEM) provides a DataWindow data cache mechanism for caching the frequently used DataWindow data. It is recommended that you enable the cache for the DataWindow objects whose data is relatively static. Any DataWindow objects whose data is fairly dynamic should remain unchecked, otherwise you will experience overhead from the caching mechanism without the true benefits of the caching.

NOTE: DataWindow Data Cache will not be effective until you fulfill all the configuration requirements that are detailed in the *DataWindow Data Cache* Section, in the *Appeon Server Configuration Guide*. Also, this feature is only supported on Windows servers. It is not supported on Unix/Linux environments.

4.3.2 Multi-thread download settings

Multi-thread downloads boost the application runtime performance. However, if there are many threads competing for the processing power of Web server, it may slow down the performance of Web server. Therefore, you should not specify an unnecessarily large number of threads in AEM. It may take some trial and error to fine-tune the performance.

4.3.3 Custom libraries download settings

If your application utilizes a customer library (e.g. DLL, OCX, etc.), you can specify in AEM how the custom library should be downloaded to the client:

- In most situations you should set the install mode to "Install automatically without asking user" or "Confirm with user, then install automatically". With these options, the custom libraries will be automatically downloaded and seamlessly installed to the Web browser.
- However, if the file size of the custom libraries is extremely large (e.g. tens of megabytes), you should set the install mode to "Install manually (no automatic installation)". With this option, Appeon does not automatically download and install the custom libraries. As such, you must distribute the custom libraries to users and your users need to install it manually.

For more details about the custom libraries download settings, refer to the *Modifying Custom Libraries Install Settings* Section in the *Appeon Server Configuration Guide*.

4.3.4 Log file settings

Once your application is fully tested and ready to move to a production environment, it is recommended to disable the AEM log functionality. Writing log files incurs disk activity, which can impact performance. Generally, the impact is small but nonetheless it will not hurt to disable this.

4.4 Internet Explorer performance settings

For optimal performance, it is recommended that the Web file caching functionality of Internet Explorer be fully utilized. This will significantly reduce the time required to load and start an application following the initial load. The configuration outlined below will ensure that you realize the best performance while safeguarding your application from becoming "stale".

STEP 1 – Open Internet Explorer and select Tools | Internet Options. Verify that the *Empty Temporary Internet Files folder when browser is closed* option is not checked under the Security section of the Advanced tab of Internet Options.

STEP 2 – Click the *Settings* button under the General tab to configure the Temporary Internet Files settings.

STEP 3 – Select the *Automatically* radio button and verify that the *Amount of disk space to use* scroll box is set to a reasonable number, such as 200 MB or more.

Now the browser is set to automatically check for newer versions of the Web application.

4.5 Web and application server performance settings

4.5.1 Sybase EAServer

The core processing of your Web application happens on the application server. As such, the better EAServer performs the better your Web application will perform. This section highlights several key performance settings you should definitely consider. You may refer to the *EAServer Performance and Tuning Guide* for details on how to extensively tune EAServer.

4.5.1.a JVM startup option

When starting EAServer, the -jvmtype switch specifies whether the client, server, or classic Java VM be used. It is recommended that you set the switch to the Java server VM.

4.5.1.b Configuring connection caches

JDBC driver used by EAServer connection cache

You should avoid using any JDBC-ODBC driver. Instead, use the Native-protocol/all-Java driver. The only exception is Sybase ASA, which provides the iAnywhere JDBC driver that actually performs remarkably well. For detailed information on configuring JDBC drivers refer to the *JDBC driver preparation* Section in the *Appeon Server Configuration Guide*.

Configuring the cache size

By default, EAServer is configured to establish 10 database connections. For applications with many several hundred or several thousands of users, this number is often too small. If not increased, it will hurt the performance of your application since your users will be waiting in a queue for an available database connection. On the other hand, if your Web deployment is few hundred users or less, you should not specify an unnecessarily large number as it will consume more server resources and possibly negatively affect performance.

The maximum pool size property, "com.sybase.jaguar.conncache.poolsize.max", defines the maximum number of connections to be held in the connection pool. The size property is generally set to 10%-20% of the maximum number of concurrent users. However, it is possible to use the FORCE option when connecting to obtain additional connections outside of the pool if none are available. Refer to the *EAServer Performance and Tuning Guide* for details on tuning the cache size.

4.5.1.c HTTP properties

You should configure the EAServer http.maxthreads and server.maxconnections properties such that it can handle the expected concurrent user load for the Web Server. If these properties are improperly configured, it may result in poor performance and possibly result in failed HTTP requests. The httpstat.dat file keeps track of cumulative hits on http objects.

Set the http.maxthreads property to the estimated average number of concurrent HTTP requests (including Servlets and any other server pages). For example, if it is expected that there will be 100 concurrent requests, set the http.maxthreads slightly higher (for example, 120). This will give you a margin of safety. Similarly, set the server.maxconnections to accommodate the average number of concurrent IIOP requests that are expected.

Another property you should pay attention to is server.maxthreads. Set this property to equal the combined value of http.maxthreads and server.maxconnections, and add 50 as a margin of safety (http.maxthreads + server.maxthreads + 50). However, if you are using an older version of PowerBuilder components with a bind thread set, you should increase this number as outlined in the *EAServer Performance Tuning Techniques* document.

Since each application and environment is unique, these are starting points that need to be monitored and adjusted for optimal results. Load and stress testing your application will help you to identify any issues prior to moving your Web application to a production environment.

5 Tuning: Excessive Server Calls

5.1 Overview

Excessive server calls in a given operation can create performance issues for that operation on slow and high-latency networks. If you are not familiar with the concept of "server calls", please refer to *Section 2.3: Impact of the Internet and slow networks* and then proceed with this section.

This section will provide four different techniques including code examples to minimize server calls and thereby optimize the performance of your PowerBuilder application for a WAN or the Internet.

1. Partition transactions utilizing stored procedures
2. Partition non-visual logic utilizing server-side non-visual objects (NVOs)
3. Eliminating Recursive Embedded SQL
4. Group multiple server calls into one "group" call with Appeon Labels

5.2 Technique #1: partitioning transactions via stored procedures

Imagine your PowerBuilder client contains the following code:

```
long ll_rows, i
decimal ldec_price, ldec_qty, ldec_amount

ll_rows = dw_1.retrieve(arg_orderid)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

if dw_1.update() < 0 then
      rollback;
      return
end if

for i = 1 to ll_rows
      ldec_price = dw_1.GetItemDecimal(i, "price")
      ldec_qty = dw_1.GetItemDecimal(i, "qty")

      if ldec_price >= 100 then
         ldec_amount = ldec_amount + ldec_price*ldec_qty
      end if
Next

ll_rows = dw_2.Retrieve(arg_orderid)
dw_2.SetItem(dw_2.GetRow(), "amount", ldec_amount)

If dw_2.update() = 1 then
      Commit;
else
      rollback;
end if
```

This is not only problematic from a runtime performance perspective since there would be numerous server calls over the WAN, but also it could result in a "long transaction" that would tie up the database resulting in poor database scalability.

The business logic and the data access logic (for saving data) are intermingled. When the first "Update( )" is submitted to the database, the related table in the database will be locked until the entire transaction is ended by the "Commit( )". The longer a transaction is the longer other clients must wait, resulting in fewer transactions per unit of time.

To improve the performance and scalability of the application, the above code can be partitioned in two steps:

1. First, move the business logic (or as much possible) outside of the transaction. In other words, the business logic should appear either before all Updates of the transaction or after Commit of the transaction. This way the transaction is not tied up while the business logic is executing.
2. Second, partition the transaction whereby all the Updates are moved into a stored procedure. The stored procedure will be executed on the database side and only return the final result. This would eliminate the multiple server calls from the multiple updates to just one server call over the WAN for saving all the data in one shot.

It is generally best to actually divide the original transaction into three segments or procedures: "Retrieve Data", "Calculate" (time-consuming logic), and "Save Data". The "Retrieve Data" procedure retrieves all required data for the calculation. This data usually would be cached in a DataWindow(s) or a DataStore(s). In the "Calculate" procedure, the data cached in DataStore will be used to perform the calculation instead of retrieving data directly from the database. The calculation result would be cached back to a DataStore and then saved to the database by the "Save Data" procedure.

Example of the new PB client code partitioned into three segments and invoking a stored procedure to perform the Updates:

```
long ll_rows, i
decimal ldec_price, ldec_qty, ldec_amount
//Retrieve data
dw_2.Retrieve(arg_orderid)
ll_rows = dw_1.retrieve(arg_orderid)
//Calculate (time-consuming logic)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

for i = 1 to ll_rows
      ldec_price = dw_1.GetItemDecimal(i, "price")
      ldec_qty = dw_1.GetItemDecimal(i, "qty")

      if ldec_price >= 100 then
         ldec_amount = ldec_amount + ldec_price*ldec_qty
      end if
Next
```

```
dw_2.SetItem(dw_2.GetRow(), "amount", ldec_amount)
//Save data
declare UpdateOrder procedure for up_UpdateOrder
   @OrderID = :arg_orderid,
   @amount = :ldec_amount;
execute UpdateOrder;
```

Example of code for the stored procedure to Update the database:

```
create procedure up_UpdateOrder(
   @orderid integer,
   @amount decimal(18, 2)
)
as
begin
    update order_detail set price = price*1.2
    where ordered = @orderid

    if @@error <> 0
    begin
        rollback
        return dba.uf_raiseerror()
    end

    update orders set amount = @amount
    where ordered = @orderid

    if @@error <> 0
    begin
        rollback
        return dba.uf_raiseerror()
    end

    commit
end
```

In summary, with the above performance optimization technique, the performance and scalability is improved since the transaction is shorter. The server call-inducing Updates are all implemented on the server-side rather than the client-side, improving the response time. Secondly, moving the business logic out of the transaction further shortens the transaction. If the business logic cannot be moved out of the transaction, one may want to consider implementing the business logic together with the transaction as a stored procedure. In summary, shorter transactions equals better scalability and faster performance.

5.3 Technique #2: partitioning non-visual logic via NVOs

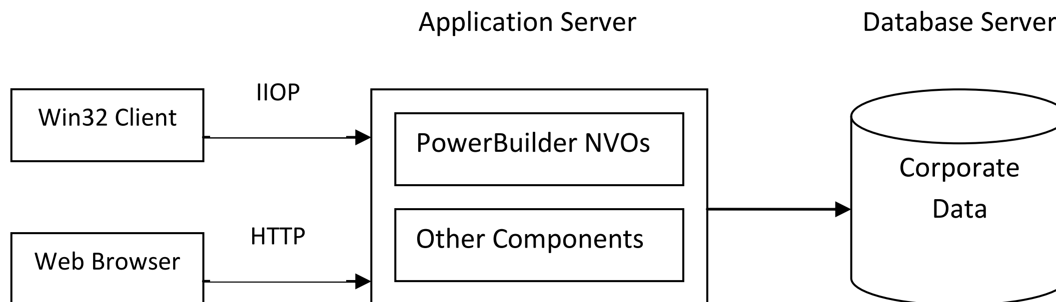
Partitioning non-visual logic and encapsulating it within PowerBuilder NVOs has been a long-time best practice among PowerBuilder developers. What's relatively new, however, is

utilizing middleware or application server, such as Sybase EAServer, Microsoft IIS, IBM WebSphere*, or BEA WebLogic*, to deploy these NVOs to the server (i.e. server-side NVOs) and invoke them from the client over IIOP or HTTP.

* *requires a Sybase plug-in that is purchased separately from Sybase or authorized resellers.*

To give you a better idea of what this looks like, the following diagram shows a very high-level architecture of PowerBuilder applications utilizing server-side NVOs when deployed as a Windows Client/Server application as well as an n-tier Web (.NET or J2EE) application with Appeon for PowerBuilder.

**Not available in Sybase distribution. refer to the Distributions section in Instroduction to Appeon.*



Deploying your non-visual logic as server-side NVOs is an excellent way to boost the performance of your application over a WAN; however, some thought must be given as what to partition. On one hand, consolidating your business logic within NVOs will significantly thin out client-side processing and move all those server call-inducing statements to the server-side running in a low-latency LAN environment. On the other hand, each invocation of the server-side NVO is a server call in itself.

If the non-visual logic you are partitioning contains multiple statements that result in server calls, then this would be a good candidate. However, if your non-visual logic does not contain any statements that would result in server calls, then by partitioning this you have not only created more work for yourself but actually added an additional server call that didn't exist before. So unfortunately it's not as simple as moving all non-visual logic to the server-side.

Imagine your PowerBuilder client contains the following code:

```
long ll_id

dw_1.Retrieve()
dw_1.SetSort("#1 A, #2 D")
dw_1.Sort()

declare order_detail cursor for
   select id from order_detail where orderid = :arg_orderid;
open order_detail;
fetch order_detail into :ll_id;

do while sqlca.sqlcode = 0
      update order_detail set price = price*1.2
      where orderid = :arg_orderid and id = :ll_id;

      if sqlca.sqlcode < 0 then
```

```
        rollback;
        return
    end if

    fetch order_detail into :ll_id;
loop
close order_detail;
commit;

dw_2.Retrieve()
dw_2.SetFilter("price >= 100")
dw_2.Filter()
```

The code highlighted in **red** above would be good candidate for partitioning as server-side NVOs while the rest of the code should remain at the PB client. After partitioning this logic, the new PB client code, which would invoke the server-side NVO, would be as follows:

```
n_order ln_order
long ll_rc

dw_1.Retrieve()
dw_1.SetSort("#1 A, #2 D")
dw_1.Sort()

ll_rc = myconnect.CreateInstance(ln_order, "PB_pkg_1/n_order")
if ll_rc = 0 then
    ln_order.of_UpdateOrderPrice(arg_orderid)
end if

dw_2.Retrieve()
dw_2.SetFilter("price >= 100")
dw_2.Filter()
```

With this technique we have reduced those numerous sever calls of the database transaction to just one single call to the NVO, and at the same time created a re-usable component that can be shared by other modules in our PowerBuilder application or shared by other applications.

5.4 Technique #3: eliminating recursive Embedded SQL

It's actually quite common to find Embedded SQL in a loop, especially Select and Insert statements. As explained previously, server calls that are recursive in nature are quite dangerous, potentially generating tremendous number of server calls. If your application requires loops or recursive functions, it would be best to replace any code resulting in server calls with code that does not.

For this technique, we will assume we have Select and Insert SQL statements in a loop. The general idea is to first create a DataWindow/DataStore using the SQL. Then replace the SQL statements contained in the loop with PowerScript modifying the DataWindow/DataStore, which does not result in server calls. If the SQL statement contained in the loop is an Insert statement, we would want to replace that with PowerScript that would insert data into the

DataWindow/DataStore. Once all the data has been inserted, then in one shot we would update the DataWindow/DataStore to the database (outside the loop), resulting in only one server call. If the SQL statement contained in the loop is a Select statement, we would retrieve data into a DataWindow/DataStore before executing the loop, and then write PowerScript in the loop to select the desired data from the DataWindow/DataStore.

The following is a code example that increases the price of a specific order by 20%, where Embedded SQL is used to update the change row-by-row (hence the loop), and then save those changes to the database:

```
long ll_id

declare order_detail cursor for
   select id from order_detail where orderid = :arg_orderid;
open order_detail;
fetch order_detail into :ll_id;

do while sqlca.sqlcode = 0
      update order_detail set price = price*1.2
      where orderid = :arg_orderid and id = :ll_id;

      if sqlca.sqlcode < 0 then
         rollback;
         return
      end if

      fetch order_detail into :ll_id;
loop
close order_detail;
commit;
```

Now we will replace the Embedded SQL with a DataWindow. Specifically, we will cache the data in a DataWindow and update the database with a single DataWindow Update, resulting in just once server call:

```
long ll_rows, i

ll_rows = dw_1.retrieve(arg_orderid)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

if dw_1.update() = 1 then
    commit;
else
    rollback;
end if
```

With this technique we have just eliminated server calls from inside the loop, reduced the number of server calls to just one, and created a data caching mechanism at the client-side that can be used to feed data to other controls of the PowerBuilder client.

5.5 Technique #4: grouping multiple server calls with Appeon Labels

Appeon for PowerBuilder provides seven "Label" functions (collectively referred to as "Appeon Labels"), which can be found in the *appeon_nvo_db_update* object in *appeon_workaround.pbl*. Appeon Labels do not execute any code or modify how your PowerBuilder application works in a Client/Server environment. Rather, when used on the Web it will notify Appeon's runtime Web libraries to handle certain database operations differently than PowerBuilder with the aim of reducing the number of server calls.

Below are details of the seven "Label" functions and specifically how they handle database operations over the Web:

Label	Function	Description
Commit/Rollback Label	of_autocommitrollback	Notifies the Appeon Web application to automatically commit or roll back the first database operation statement after the label.
Commit Label	of_autocommit	Notifies the Appeon Web application to automatically commit the first database operation.
Rollback Label	of_autorollback	Notifies the Appeon Web application to automatically roll back the first database operation statement if the operation fails.
Queue Labels (Consists of Start Queue Label and Commit Queue Label)	of_startqueue of_commitqueue	These two labels must be used in pairs. They notify the Appeon Web application not to commit database operations after the Start Queue Label until the Commit Queue Label is called (and unless an Appeon Immediate Call Label is called).
Immediate Call	of_imdcall	Notifies the Appeon Web application to immediately commit a database operation.
Update Label	of_update	It is used to reduce the number of interactions with the server caused by "interrelated updates". "Interrelated updates" usually occurs when the update result of one DataWindow determines whether another DataWindow should be updated.

Appeon Commit/Rollback Label

The Appeon Commit/Rollback Label (of_autocommitrollback) notifies Appeon to automatically commit or roll back operations to the database after updating or inserting.

For example:

```
gnv_appeonDbLabel.of_AutoCommitRollback()
```

```
update tab_a ……
if sqlca.sqlcode=0 then
      …… //code independent of database opertaions
      commit;
      ……
else
      ……
      rollback;
      ……
endif
```

Appeon Commit Label

The Appeon Commit Label (of_autocommit) notifies Appeon to commit operations to database after updating and inserting.

For example:

```
gnv_appeonDbLabel.of_AutoCommit()
update tab_a ……
```

Appeon Queue Labels

There are two Appeon Queue Labels, the Start Queue Label (of_startqueue) and the Commit Queue Label (of_commitqueue). These two labels must be used in pairs. They notify Appeon not to commit database operations after the Start Queue Label until the Commit Queue Label is called (and unless an Appeon Immediate Call Label is called).

For example:

```
gnv_appeonDbLabel.of_StartQueue()
dw_1.retrieve(arg1,arg2)
dw_2.retrieve(arg3,arg2)
…
dw_3.retrieve(arg4)
gnv_appeonDbLabel.of_CommitQueue()
```

Appeon Immediate Call Label

The Appeon Immediate Call Label (of_imdcall) is used between the Appeon Start Queue Label and Appeon Commit Queue Label, when the return value of an operation that is called after the Appeon Start Queue Label determines the subsequent business logic, for example, the return value is used in a CASE or IF…THEN expression.

For example:

```
gnv_appeonDbLabel.of_StartQueue()
dw_1.retrieve()
gnv_appeonDbLabel.of_ImdCall()
select … into :var_1,:var_2……
if var_1>0 then
      para = "ok"
```

```
else
      para = "false"
end if
dw_2.retrieve(para)
gnv_appeonDbLabel.of_CommitQueue()
```

Appeon Update Label

The Appeon Update Label (of_update) is used to reduce the number of interactions with the server caused by "interrelated updates". "Interrelated updates" usually occurs when the update result of one DataWindow determines whether another DataWindow should be updated.

The following example shows how Appeon uses the Update Label to reduce client-server interactions:

Example of interrelated updates:

```
if dw_1.Update()=1 then
    if dw_2.Update()=1 then
        commit;
        Messagebox("Success","Update success!")
    else
        rollback;
        Messagebox("Failure","Update all failure!")
    end if
else
    rollback;
    Messagebox("Failure","Update dw_1 failure!")
End if
```

Use the Appeon Update Label to rewrite the example:

```
l_rtn = gnv_appeonDb.of_Update(dw_1,dw_2)

if l_rtn=1 then

   Messagebox("Success","Update success!")

elseif l_rtn= -102 then

   Messagebox("Failure","Update all failure!")

Else

    Messagebox("Failure","Update dw_1 failure!")

End if
```

Script defined in the Update Label associated function, of_Update(dw_1,dw_2):

```
if dw_1.Update()=1 then
    if dw_2.Update()=1 then
        commit;
```

```
        return 1
    else
        rollback;
        return -102
    end if
else
rollback;
return -101
end if
```

The more database operations utilize Appeon Labels, the faster the performance will be. For PowerBuilder applications deployed to the Web with Appeon for PowerBuilder, in many cases you will achieve acceptable runtime performance simply by utilizing this technique. The reason is that there are a number of features built into Appeon's infrastructure framework that automatically boost the performance of PowerBuilder applications over the Web. The performance boosting features are discussed in *Section* 2.2: *Automatic performance boosting*.

6 Tuning: Heavy Client

6.1 Overview

If you find your Appeon Web application performs poorly even when run in a local environment, then chances are the JavaScript interpreter of the Web browser is slowing you down. Generally, a newer or more high-power computer will clear up many situations. But if it is not an option to utilize a late model computer or if the performance issue still persists, then you would want to consider optimizing your PowerBuilder code to make it more efficient when running in the Web browser. This section provides several different techniques for dealing with several specific types of inefficient PowerBuilder coding practices.

6.2 Technique #1: thin-out "heavy" Windows

Redesign the navigation strategy to present a lighter-weight Client user interface. Specifically, you would want to focus on reduce the number of DataWindows and DropDownDatawindows in a particular window or tab page. In many situations, the DataWindows in a Window can be spread out across multiple Windows or tabs, thereby reducing the "weight" of the Window. It may be possible to rework your DropDownDataWindows as DropDownListBoxes. By thinning out the UI, it will not only make the Window run faster but your users will not be overwhelmed by so much data.

6.3 Technique #2: thin-out "heavy" UI logic

This section is broken down into several subsections, utilizing the same technique to deal with various types of "heavy" UI logic.

6.3.1 Manipulating the UI in loops

Excessive and uncessary loops have a negative impact on performance. Some PowerBuilder code will trigger your Appeon Web application to redraw visual objects, such as DataWindows, controls, etc. If such functions are put into a loop, it will redraw the visual objects numerous times and therefore negatively affecting performance.

Appeon recommends the following:

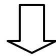
- Do not put functions that operate on DataWindow rows into a loop.
- Avoid placing functions that result in the repaint of visual control(s) into a loop; otherwise, the visual control(s) will be repainted many times while the loop is executed.
- Use the Find function for DataWindow search instead of using the loop statement.

The following is an example:

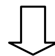
```
long ll_row
String ls_expression
String ls_Name
ls_Name = "Mike"
For ll_row = 1 To dw_1.RowCount()
      ls_expression =
         dw_1.GetItemString(ll_row,"name")
```

```
      If ls_expression = ls_Name Then

         Exit
      End If
Next
```



```
long     ll_row
Long     ll_rowcount
String        ls_expression
String        ls_Name
ls_Name = "Mike"
ll_rowcount = dw_1.RowCount()
For ll_row = 1 To ll_rowcount
      ls_expression = +
      dw_1.GetItemString(ll_row,"name")
      If ls_expression = ls_Name Then
                    ...
      Exit
End If
Next
```



```
Long     ll_row
Long     ll_rowcount
String                ls_expression
String        ls_Name
ls_Name = "Mike"
ll_rowcount = dw_1.RowCount()
ls_expression = "name = '" + ls_Name + "'"
ll_row = dw_1.Find(ls_expression,1,ll_rowcount)
...
```

6.3.2 Triggering events repeatedly

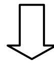
Frequent triggering of events such as Timer, MouseMove, and SelectionChanging slows down performance. For example, once a Timer event is triggered, it occurs repeatedly at a specified interval that can be set to even milliseconds (1/1000 of a second). Minimize the usage of events with high repetition such as Timer, MouseMove, SelectionChanging, GetFocus, LoseFocus, Activate, and Deactivate.

6.3.3 Performing single repetitive tasks

Use batch operations instead of performing a single operation many times. For example, the execution of the following "batch" code is two to three times faster than the original code.

```
For I = 1 To 100
      dw_1.SetItem(ll_i,+
                          ”name”,+
```

```
                              dw_2.GetItemString(ll_i,"name"))
                        …
Next
```



```
dw_1.RowsCopy(1,100,Primary!,dw_2,1,100,Primary!)
```

6.3.4 Initializing “heavy” tabs

For windows containing Tab controls, if the Tab control contains more than five tab pages or the initialization of each tab page is complex, Appeon recommends you use the CreateOnDemand method of Tab control to improve the runtime performance of these windows.

When CreateOnDemand is enabled, only the current tab page that is created will be initiated. The initialization of the hidden tab pages takes place only when they are selected. Therefore the window will operate more efficiently.

Please keep in mind that as you make this change you may also need to modify other code of the Tab control. For example, if the current tab page uses data from another page, you need to:

1. Move the script that is used for obtaining data, to the SelectionChanged event.
2. Add a condition to validate whether the tab page carrying the data has been initiated. If initiated, the current tab page will successfully obtain the data; if not initiated, the user must select the tab page for initialization purposes, and the current tab page will successfully obtain the data.

6.3.5 Using ShareData or RowsCopy/RowsMove for data synchronization

The following PowerBuilder functions can synchronize data between DataWindows: ShareData, RowsCopy/RowsMove, Object.Data, and SetItem. ShareData is the fastest and it is recommended to use it whenever you need to synchronize data between DataWindows. SetItem is the slowest and should be avoided as much as possible. If ShareData cannot be or should not be used for some reason, then consider using RowsCopy/RowsMove followed by Object.Data.

6.3.6 Using computed fields

Computed columns involve a lot of recalculation in many situations; for example, when a column is deleted, added, or renamed. This recalculation is a process-intensive task, which negatively impacts performance and can be worked around. Therefore, Appeon recommends the following:

- Avoid using computed columns in detail bands. Instead, add expressions in the SQL statements for getting specific data.
- Avoid embedding a computed column in an existing computed column.
- If a computed column is “*Text: Sum or Expression*”, it is recommended that you divide the column into two columns: an edit style column with the “*Text*”, and a computed column with “*Sum or Expression*”.

6.3.7 Using DataWindow expressions

Generally speaking, DataWindow expressions will slow-down the initial display or subsequent refresh of DataWindows. As such, Appeon recommend you reduce the usage of DataWindow expressions if possible, especially in the following situations:

- Avoid using DataWindow expressions for computing and setting column properties.
- Avoid setting sort and filter criteria directly for a DataWindow object. Instead, write the sort and filter criteria in the SQL statement of the DataWindow object. As noted previously, it is faster to use SQL statements than DataWindow functionality.

6.3.8 Using complex filters

Filters are considered "complex" if the filter criteria contain one or more expressions that call to one or more functions. It is recommended that you not use complex filtering on a DataWindow, especially on a DataWindow that has large amounts of data.

6.3.9 Using RowsFocusChanging/RowsFocusChanged events

The DataWindow RowsFocusChanging and RowsFocusChanged events can be triggered under many situations, especially when a DataWindow retrieves data. Since data is usually automatically retrieved into DataWindows when a Window is opened, if a lot of code is written into the RowsFocusChanging and RowsFocusChanged events, it will significantly prolong the time it takes to open the Window and display the DataWindow. Appeon recommends that you do not write code into RowsFocusChanging and RowsFocusChanged events unless it is necessary.

6.4 Technique #3: offload "heavy" non-visual logic

Instead of trying to write "heavy" logic more efficiently or avoiding use of "heavy" logic, the simplest way is just to offload all that "heavy" logic to the application server, which is designed to handle the most daunting tasks. The only catch is that only non-visual logic can be run at the application server.

The following types of non-visual logic can be encapsulated in PowerBuilder NVOs and deployed to the application server to eliminate "heavy" logic from the Web browser:

- Complex non-visual events and functions, especially non-visual events and functions that contain dynamic SQL, Cursor statements, Stored Procedure calls, and other SQL statements.
- Validation of updated data.
- A series of data computations or a complex data computation.

7 Tuning: Large Data Transmissions

7.1 Overview

Suppose you have worked hard to make an application Web-ready using Appeon, and, using your test data, it seemed to perform acceptably. Then, when your users provide "live" test data in realistic volumes, you discover that the application takes a long time to load, and worse, a long time to respond to your user's input. What to do?

Well first you should confirm that your issue is not being caused by excessive server calls (see *Chapter 5: Tuning: Excessive Server Calls).* The reason is that majority of the time, PowerBuilder applications are coded such that as additional rows of data are retrieved logic is executed to validate, manipulate, or otherwise handle the data, which can result in server calls. As such, the more rows of data are retrieved the more server calls are made.

Once you are certain the slow-down is not caused by excessive server calls then you can consider reducing the size of data transmission. So what can do practically? Well at a high-level there are several techniques you can employ:

- The first and most popular is staging the data retrieval into manageable increments. For example, you can expose a Next button, and have the application respond to this button click by getting the next logical segment of the result set just like typical Websites or Web applications. *Section 7.2: Technique #1: retrieving data incrementally* gives you instructions on how to achieve this.
- Another technique is to create multiple smaller "specific" views rather than one larger "general" view. Consider adding SQL WHERE clauses based on more search criteria, thus retrieving only the amount of data that is absolutely necessary for a particular view of interest.
- If you have a choice between reducing the number of rows retrieved, and reducing the number of columns, note that a small reduction in columns (described below in *Section 7.3: Technique #2: minimizing excessive number of columns*) can improve performance to an even greater extent than a reduction in rows. This is because most of the time, loops, whether in the application code or in the virtual machine, visit columns first and then rows.

Anything you do to reduce the size of the result set in one way or another can only improve performance and possibly improve usability of your application as well.

7.2 Technique #1: retrieving data incrementally

7.2.1 For Oracle database server

Oracle includes a pseudo-column called ROWNUM which allows you to generate a list of sequential numbers based on ordinal row. If your application uses Oracle database, apply your Oracle skills and ROWNUM to limit the number of returned rows. For example, this query selects the TOP 10 rows from a table:

```
SELECT *
FROM   (SELECT * FROM my_table ORDER BY col_name_1)
WHERE  ROWNUM BETWEEN 1 AND 10;
```

You can impose a NEXT button to the DataWindow. In the Clicked event of the NEXT button, the query changes with ROWNUM increments by 10. Therefore, when the NEXT button is clicked, the DataWindow displays next 10 rows.

7.2.2 For all other database servers

If your application uses a non-Oracle database (for example, Microsoft SQL server) you can use the following SQL syntax to limit the number of returned rows to the DataWindow:

```
SELECT TOP 10 *
FROM my_table
WHERE Table.primary_key > = bottom
ORDER BY Table.primary_key;
```

"bottom" is a variable that contains the row number of the first row you want to retrieve, where rows are ordered by the primary key for the table. Before retrieving the first page of data, "bottom" should be set to a value smaller than any primary key value in the table.

Based on this SQL statement, you can implement Next and Previous buttons for the DataWindow. Their Clicked events increment or decrement the bottom variable so that its value matches the primary key value in the first row you want to retrieve then execute the above SQL statement.

7.3 Technique #2: minimizing excessive number of columns

As the number of rows in the result set increased, the number of columns will cause greater degradation on performance, especially for nested loops in your application which process rows in the outer loop, and columns in the inner loop. Sometimes the excessive number of columns is intentional and other times it is unintentional.

A sign of unintentionally excessive columns would be the SQL syntax Select * From: consider modifying this syntax to Select *fieldList* From, where *fieldList* is the comma-separated list of all, and only, those fields your application will actually need. The performance of the SQL syntax using asterisk will be automatically degraded any time your database administrator modifies the database design by adding columns.

A sign of intentionally excessive columns is simply a long list of columns in your SQL Select statement. Consider analyzing your actual needs to make certain all columns are necessary. It may be possible to request certain columns (needed only in exceptional circumstances) in a separate SQL operation. Please keep in mind if the Visible property of a column is set to zero (the control is not visible), even though the Column cannot be seen, it is still impacting performance.

8 Conclusion

PowerBuilder applications that perform well today in your local network may not perform well in a distributed architecture tomorrow. Likewise, typical PowerBuilder development practices may not be suitable for a distributed architecture. The several techniques outlined in this guide are intended to steer you in general directions. It is recommended to extrapolate from these examples and apply to your particular situation. Please keep in mind that excessive server calls is the single biggest culprit of performance issues over the Internet, which is a relatively high latency connection.

Purchasing expensive network connectivity and faster hardware can make up for suboptimal code. Sometimes the cost of doing this is less than the cost of optimizing the code. If you do take this route, keep in mind that a low-latency network connection is generally the key rather than a high-bandwidth connection. Reason being, for most PowerBuilder applications and deployments, it is the network latency that kills the runtime performance not bandwidth limitations.

Index

1

10X data compression, 7

A

about this book, 1
all other database servers tuning, 27
Appeon 6.0 performance
  Automatic performance boosting, 4
  Expected performance level, 4
  Impact of heavy client-side logic, 6
  impact of large data transmission, 7
  Impact of the Internet and slow networks, 5
Appeon 6.0 Performance, 4
Appeon Developer performance settings, 9
Appeon Enterprise Manager performance settings, 9
  custom libraries download settings, 10
  log file settings, 10
Appeon Enterprise Manager performance settings
  DataWindow Data Cache, 9
Appeon Enterprise Manager performance settings
  multi-thread download settings, 9
application server performance settings
  Sybase EAServer, 10
audience, 1
automatic performance boosting, 4

C

Configuring connection caches, 11
custom libraries download settings, 10

D

DataWindow Data Cache, 9
DataWindow, performance tuning
  RowsFocusChanging and RowsFocusChanged events, minimizing code, 25

E

EAServer JVM startup option, settings, EAServer performance, 11
EAServer performance, configuring
  connection caches, 11
  EAServer JVM startup option, 11
  http properties, settings, 11
  JDBC driver used by EAServer connection cache, settings, 11
eliminating recursive embedded SQL, 16
expected performance level, 4

G

grouping multiple server calls with Appeon Labels, 18

H

heavy client-side logic, 6
heavy windows, thin-out DataWindows, reducing, 22
how to use this book, 1
Http properties, settings, EAServer performance, 11

I

identifying performance bottlenecks, 8
Identifying Performance Bottlenecks
  heavy window report, 8
if you need help, 2
Impact of heavy client-side logic, 6
Impact of the Internet, 5
Impact of the slow networks, 5
initializing, 24
Internet Explorer configuration
  caching recommendation
    disk space to use, 10
    temporary Internet files folder, Advanced tab, 10
    temporary Internet files, General tab, 10
Internet Explorer performance settings, 10

J

JDBC driver used by EAServer connection cache, settings, EAServer performance, 11

L

large data transmission, 7
log file settings, 10

M

manipulating the UI in loops, 22
minimizing excessive number of columns, 27
multi-thread download settings, 9

O

offload, 25
Oracle database server tuning, 26

P

partitioning transactions, 12
performance bottlenecks overview, 8
performance-boosters
  10X Web File Compression, 4, 9
performance-related settings
  Web and application server performance settings, 10
performance-related settings, 9
  Appeon Developer performance settings, 9
  Appeon Enterprise Manager performance settings, 9
  Internet Explorer performance settings, 10
  overview, 9
performing single repetitive tasks, 23

R

readers, 1
related documents, 1
retrieving data incrementally
  all other database servers, 27
  Oracle database server, 26
retrieving data incrementally, 26
RowsFocusChanging and RowsFocusChanged events, minimizing code, 25

T

thin-out, 22
triggering events repeatedly, 23
tuning
  excessive server calls
    partitioning non-visual logic via NVOs, 14
  heavy client, 22
    overview, 22
    thin-out, 22
  large data transmissions
    minimizing excessive number of columns, 27
    overview, 26
    retrieving data incrementally, 26
  large data transmissions, 26
  offload, 25
  thin-out, 22, 23, 24, 25
tuning
  excessive server calls, 12
tuning
  excessive server calls
    eliminating recursive embedded SQL, 16
tuning
  excessive server calls
    grouping multiple server calls with Appeon Labels, 18
tuning excessive server calls
  partitioning transactions via stored procedures, 12
tuning excessive server calls
  overview, 12
tuning partitioning non-visual logic, 14
tuning the cache size, 11

U

using complex filters, 25
using computed fields, 24, 25
using RowsFocusChanging/RowsFocusChanged events, 25
using ShareData or RowsCopy/RowsMove for data synchronization, 24

W

Web and application server performance settings, 10