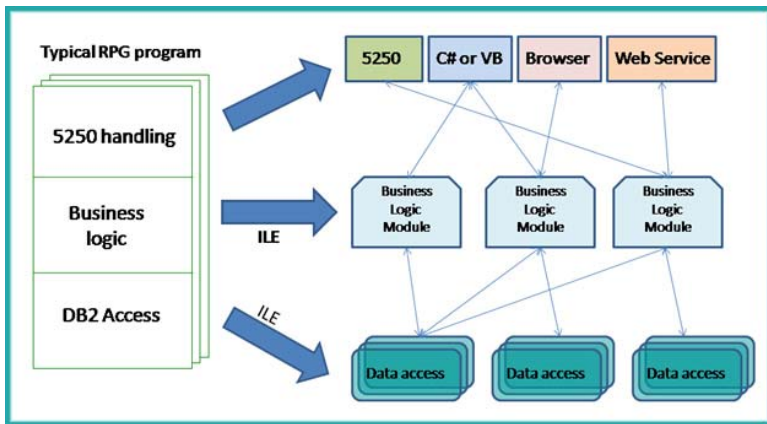


## APPLICATION MODERNIZATION VIA DATA MODERNIZATION

One of the most common challenges in iSeries shops is the need to modernize legacy applications. There are dozens of approaches and many good tools available to achieve the goal of making these tried-and-true applications easier to maintain and to replace their 5250 screens with a graphical interface. The challenge is to determine what the most cost effective and efficient strategy is – and based on that deciding where to start. All of this of course, must be done in the context of running the business and mitigating risk as changes are introduced.

While starting the modernization process with the user interface is a popular approach, you can also employ a bottom-up strategy by starting with the database. In fact it may make sense to split responsibilities and do work in parallel by having one team start at the interface and work down while another team starts at the database and works up. Here is a graphical representation of the process:

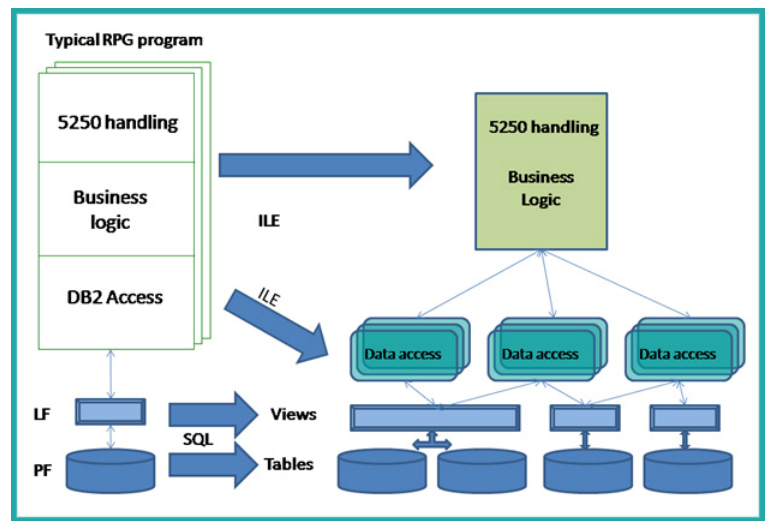


Here are the fundamental themes in this chart:

1. Break monolithic programs that contain 5250 handling, business logic, and database access into three distinct and modular layers.
2. Convert from old-style RPG to ILE RPG so that breaking business logic into small, reusable pieces is supported by the language.

3. Isolate data access logic so that the impact of database changes is minimized to allow greater flexibility as the business needs change and information evolves.

Another way to do modernization is to focus on the database layer and work up. This would result in an intermediate picture that looks something like this:



The themes in this approach are:

1. Reduce risk and/or effort by focusing the modernization effort at the lowest layers and leaving a majority of the business logic unchanged.
2. Isolate database access to a small number of stand-alone routines so that evolution at the database level does not require programs to be modified.
3. Begin to transition from native RPG data access methods to SQL.

### Inside this issue:

Application Modernization via Data Modernization	1
Server Mode Mysteries	2
CTO Memo	3
Controlling Temporary Storage Consumption	5

(Continued on page 3)



# Server Mode MYSTERIES

Have you ever wondered what QSQRVR jobs do? Maybe you already know what they do but are having a difficult time managing them? Hopefully this article will clear up some of the confusion.

QSQRVR jobs were invented by IBM to support what is called "SQL Server" processing mode. Do not confuse these words with Microsoft's SQL Server database. This type of processing is unique to the iSeries system. Essentially this processing style allows a job to delegate the responsibility of running an SQL statement to a secondary process and return the results. In other words, a job using SQL Server mode does not execute a query within its own job but lets the QSQRVR pre-start job do the work.

## Why was SQL Server mode needed?

SQL server mode is necessary for applications that have multiple threads within a single job. Java applications in particular are generally multi-threaded and can have database connections open in each of those threads. For a variety of reasons, including the fact that a job can only have one active transaction per activation group, server mode was needed.

## Who uses Server mode?

Server mode is used by a growing number of functions. By default, the native JDBC driver on the IBM i uses this approach to execute SQL statements. Applications using the Call Level Interface (CLI) can optionally use this method, and a majority of them do. Oracle's EnterpriseOne ERP suite is a good example. IBM's new DB2 Web Query product also uses it.

## What are the benefits of Server mode?

There are several nice benefits of Server mode. The first, as already mentioned, is that it allows multiple transactions (i.e. commit cycles) to be utilized within an application. This provides much-needed flexibility when performing complex and inter-related database operations.

Second, Server mode allows work to be done in parallel because if more than one QSQRVR job is executing SQL statements on behalf of the controlling jobs more work will get done. One could argue that there is also a performance penalty to be paid for the communication overhead between jobs and you would be correct. IBM, however has a relatively efficient mechanism to pass information back and forth between the jobs so the cost is fairly small. Finally, a well written application with



The development team here at Centerfield is hard at work. We've started development on the next version of HomeRun. While the release date is still being determined, the next version will contain some pretty neat functions that I'm confident you'll love. Among the themes of the next release are:

- Increased automation of database management tasks that are very time-consuming, technically difficult, or both.
- Advanced management reports that quantify the value of advice provided by HomeRun after doing in-depth analysis on your database objects.
- Real-time insight into how your database is being used as applications and jobs run right now. Expert advice will also

be generated immediately after HomeRun is installed so you don't have to wait for the weekly scheduled analysis to run.

In the meantime if you have DB2 or SQL-related challenges that you'd like to solve let us know about those – you never know, we might already be able to fix them!

Best Regards,

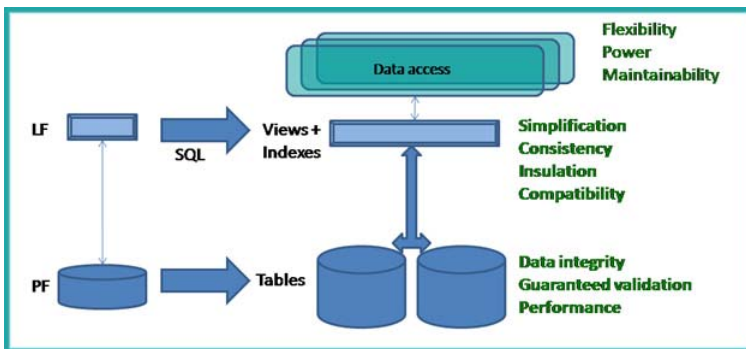
Mark L Holm  
Chief Technology Officer  
Centerfield Technology, Inc.

*(Application Modernization via data Modernization — Continued from page 1)*

4. Move from DDS to SQL at the data definition and data storage layers.

This approach has several benefits that include reduced program maintenance, higher programmer productivity by leveraging the power of SQL, faster performance in many situations, and the availability of SQL skills.

Let's take this picture down one more level before we discuss the conversion process in detail:



This graphic explains not only the process to migrate to SQL-based access but the benefits of doing so. Let's cover each of the benefits and the process to achieve that benefit.

**Flexibility:** When native database requests are written in RPG they are generally fixed. Traditionally, if the RPG programmer wanted to change the columns and rows selected at runtime, they would use the OPNQRYF command but that would still require a significant amount of code. Database queries can be dynamically built and executed with SQL in a very convenient manner.

**Power:** SQL has the ability to select records of interest based on very complex criteria. It can also manipulate and combine

existing data to dynamically create new information or calculated results thus removing that responsibility from the programmer. While anything could be done with the right combination of logical files and RPG coding, it is much simpler to push the job of doing that work to SQL.

**Maintainability:** Database operations in native RPG are “hardwired” to the format of the file they are accessing. SQL, on the other hand, allows the format of the underlying file to change without necessarily being impacted by those changes. For example, a field can be added, removed, or changed without always forcing the program to be recompiled if that field is not referenced by the SQL statement.

**Simplification:** SQL views, and before that DDS logical files, were a good way to hide the complexity of a database from both end-users and programmers. This simplification makes it much easier for everyone to understand and use information stored in complex file structures. SQL views go above and beyond what can be done in DDS. Furthermore, views can now be defined so they can replace those logical files while preserving the file formats thus eliminating impact to existing programs.

**Consistency:** Because SQL views can join tables together, can calculate results, can aggregate data, and can map cryptic values to readable ones, they provide a consistent view of business data to an end-user. For example, rather than have different people write reports that may calculate profit margin differently it could be standardized in an SQL view that all reports build upon.

**Insulation:** Much like the data access programs or routines insulate business logic from database changes, SQL views do the same thing for the data access routines. If those programs access views rather than the physical table directly, the table can change without necessarily causing the data access programs to change.

*(Continued on page 4)*

(Server Mode Mysteries — Continued from page 2)

many users can consume fewer resources in aggregate since the QSQRVR jobs can service more than one “host” which reduces the system-wide overhead for memory and cpu resources.

The final benefit is that limits to connection management are lifted when using Server mode. Jobs executing SQL in-line can only have one connection to the database and while Server mode allows multiple.

### What are the downsides to Server mode?

The primary issue with QSQRVR jobs is that they can consume a lot of resources (as any job running SQL can). While this could be considered business-as-usual, what makes it more difficult is that it isn't always obvious WHO the QSQRVR job is doing work for. This can be problematic if the application is written to use a generic user profile. This eliminates the most obvious link back to the requesting job.

Recognizing this problem, IBM has started to put a message in the joblog of the process using Server mode in order to allow administrators to connect the requester with a specific QSQRVR job. When a connection is established with a QSQRVR job, an SQL7908 message is sent that contains the full job name of the job running its SQL statements. Unfortunately, this isn't very helpful because performance or functional problems typically occur in the QSQRVR jobs and it does not have a link back to the “owner” of the work.

If the job using Server mode is running under a distinct user profile, then it is somewhat easier to link the QSQRVR job back to a particular requester. By doing a WRKOBJCK on the user profile of the QSQRVR job's current user profile all of the jobs running under that profile can be seen. [Alternatively, the WRKACTJOB screens show the current profile of all jobs by default.]

### What are the mechanics of using Server mode?

SQL Server mode must be explicitly requested by the software that wants to use this method. Most of the time the request is done by the application (e.g. Oracle EnterpriseOne) or middleware (e.g. IBM's native JDBC driver). If you want to write your own applications that use CLI, you must use the SQLSetEnvAttr() function to tell DB2 to enable Server mode. Alternatively, you could use the QWTCHGJOB (the Change Job work management API) to do the same thing.

Once enabled, SQL requests will be passed to one or more QSQRVR jobs for execution. The good news is that the process is transparent to the application. In other words, the use of SQL Server mode will not require changes to the application (other than code to enable that mode in the first place).

Because QSQRVR jobs run by default in the QSYSWRK subsystem, you may not get the same performance characteristics if the memory pool(s) allocated to that subsystem are small in comparison with the requester's pools. Alternatively, the QSYSWRK pools can be increased in size to make the requests run faster than if they were run by the job in-line.

### What about authority and similar job attributes?

In order to function correctly, any work done by a QSQRVR job must be performed with the proper user authority. In order to do that, the QSQRVR job will dynamically switch user profiles so that all object authority is honored. Furthermore, other job attributes like priority (technically priority is inherited from the invoking thread), language settings, formatting options and many others are also inherited by QSQRVR.

### Summary

SQL Server mode is becoming more and more common on the iSeries. To better manage these types of jobs it is important to understand how they work. Here are some tips for making management easier:

- If possible, use Server mode in jobs that run with a unique user profile instead of a generic user.
- Look at the SQL7908 message in the requesting job to identify which QSQRVR jobs are being used.
- Tune the memory pools in the QSYSWRK subsystem to ensure they are given adequate resources to run an SQL workload.
- Adjust the number of QSQRVR jobs with the CHGPJE command so that you create as many of them as you need during a typical workday in order to minimize delays that occur when you run out of available server jobs.

(Application Modernization via data Modernization — Continued from page 3)

**Data integrity:** SQL tables by definition do basic data validation when data is added or updated. For example, the historical problem of blanks being inserted into zoned decimal columns is avoided because those invalid values are not allowed. Furthermore, by default SQL tables are journaled which provides a needed level of protection from power outages and similar catastrophic failures.

**Guaranteed validation:** Besides basic data integrity enforcement, SQL tables support advanced functions like constraint definitions which provide power and flexibility to enforce just about any business rule. Embedding these rules into the database has several huge benefits. The first is that the validation will happen no matter what tool is used to change the data. Secondly, it helps reduce the size of programs that may have had duplicate code (or worse yet code that wasn't consistent) to enforce these rules.

**Performance:** The fact that tables do data integrity checking as data is inserted or changed allows DB2 to take greater liberties when the data is retrieved from the file – in other words less data checking is done when the information is read. This means that the performance of SQL tables will be slightly better than DDS physical files because data is generally read many more times than it is changed.

As you can see there are many benefits of tackling the modernization “problem” from the bottom-up. It also has many advantages to an already over-busy IT shop because it can be done with less time, effort, and risk.

# CONTROLLING TEMPORARY STORAGE CONSUMPTION

When you perform a WRKSYSSTS (Work with System Status) command on your system, you may have noticed two metrics named "Current unprotect used" and "Maximum unprotect".

```
Current unprotect used      :      3169 M
Maximum unprotect          :      3303 M
```

These metrics pertain to current and peak temporary storage consumption. Normally they don't warrant a second look.

On rare occasions however, you may see the temporary storage value skyrocket. If you have system monitoring tools in house, this will raise panic alarms and set your system operators into action. If this condition is not detected and corrected in a timely fashion, in most extreme cases it may actually cause the system to run out of storage in the system ASP (ASP number 1) and induce an automatic IPL to prevent a catastrophic failure.

In this article I will talk about the usual suspects for this error condition and outline number of methods you can employ to prevent the unscheduled system IPL.

## What is temporary storage

Temporary storage is the unprotected storage allocated by the OS in the system ASP whose addressability will be lost when an IPL occurs. Some examples of this storage are open data paths, compiler work areas, programs loaded into the activation group and the associated variables, heap space (Java as well as other HLLs, i.e. malloc, calloc), query optimizer temporary objects like hash tables, sorts, temporary indexes and so on. This is by no means an exhaustive list.

Temporary storage does **not** include objects (i.e. files, user spaces, user indexes etc.) created in QTEMP libraries for jobs on the system. Since QTEMP is a library (context), it is treated as a protected (permanent) storage by the OS even though it is transient in nature from an end user and developer standpoint.

## First protective measure – get current on temporary storage leak PTFs

Extreme usage of temporary space to the point of running out of storage in system ASP always indicates an error condition. Since great majority of temporary objects are created by the OS itself, it makes sense that the OS is a frequent culprit behind temporary storage leaks. IBM recognizes the severity of the issue temporary storage leaks can cause to the end-user, so they provide a special website that lists all of the temporary storage PTFs, categorized by OS release version:

[http://www-912.ibm.com/s\\_dir/slkbase.NSF/DocNumber/23341308](http://www-912.ibm.com/s_dir/slkbase.NSF/DocNumber/23341308)

As a first protective measure, make sure you are always current on temporary storage PTFs.

IBM offers a subscription service, where you can be notified of new temporary disk space issues, so you don't have an excuse of not always being current with these type of fixes.

**NOTE:**  
*Limiting temporary storage consumption causes the OS to abruptly terminate (kill) the job that reached the designated limit. If the application running into this limit is not coded to handle the unexpected termination of such nature, you may see unpredictable application problems. Hence, use caution when implementing any of the methods outlined in this article.*

(Continued on page 6)

(Controlling Temporary Storage Consumption—Continued from page 5)

To subscribe, follow this link:

[http://www.ibm.com/systems/support/myview/subscription/css.wss/subscriptions?methodName=createNewSubscription&css\\_key=i027&brandid=5000027](http://www.ibm.com/systems/support/myview/subscription/css.wss/subscriptions?methodName=createNewSubscription&css_key=i027&brandid=5000027)

### **Second protective measure – implement sound indexing strategy**

Being that Centerfield is most well known for database and SQL tuning expertise, one of the most common culprits behind excessive temporary storage consumption we hear about is the SQL query engine. In some occasions it is a known issue and there is an IBM PTF that corrects it.

In more frequent occasions, the root cause behind excessive temporary storage use is simply a lack of SQL indexes in the database. At the risk of oversimplifying the artificial intelligence of the query optimizer, let me say that it has only two choices to implement your query using permanent objects on the system: a table scan or index implementation. If a database architect and database administrator have not built appropriate SQL indexes (or keyed LFs) to offer an alternative implementation option as well as a real-time source of database statistics, the query optimizer has no choice but to use a table scan to find the rows that fit SQL query's selection/ordering/grouping criteria. Once it scans the necessary rows, it'll resort to a variety of sophisticated techniques to further refine the data you requested (i.e. hash table, sorted list, distinct sorted list, temporary list, values list, row number list, bitmap, temporary index, buffer, queue...). All of these objects require temporary disk space which translates to real auxiliary storage use in ASP 1.

The most common example of this scenario is performing a join between two large files over fields that are not keyed anywhere (no primary keys, SQL indexes, keyed LFs, unique constraints, foreign key constraints – in short, no keyed access path over a join column). In that scenario, DB2 has no choice but to employ a method called Cartesian Product Join. This means that the product of two files each with one million rows in them could **potentially** produce a temporary result with  $1M * 1M = 1$  Trillion rows.

I say potentially, since the query optimizer is very smart if at all possible it'll avoid that implementation. If left with no choice however, it'll do what **you've asked it to do**.

You can imagine what can happen if you have multiple Cartesian product join queries running at the same time – temporary storage consumption skyrockets.

As a second protective measure, implement a sound indexing strategy on your database.

The following links are excellent starting points on your way to indexing nirvana:

Indexing and Statistics Strategy for IBM i  
<http://www-03.ibm.com/servers/enable/site/bi/strategy/index.html>

OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4  
<http://www.redbooks.ibm.com/abstracts/sg247326.html>

Also, keep in mind free of charge iSeries Navigator tooling as well as best of breed database and SQL performance tuning tooling offered by Centerfield Technology, your author's benevolent employer.

### **Third protective measure – control and limit temporary storage consumption**

A large number of our customers run multibillion dollar businesses and their systems have to be up 24/7. They cannot afford an unscheduled IPL due to excessive temporary storage consumption. They follow preventive measures outlined in steps one and two, but they want to be absolutely certain they won't miss some wayward JVM garbage collector leak, communication stack leak or a poorly written application program leaking heap space. In other words, they want to control and limit the maximum amount of temporary storage anyone on the system can use.

There are a variety of approaches you can employ to effect a limitation on temporary storage use, and I'll list the ones I am aware of.

- 1) Create or change an existing work management class object (\*CLS) and set the MAXTMPSTG value to something other than the default value of \*NOMAX.

For example, one of the most common complaints I hear about is temporary storage use of the QZDASOINIT jobs (ODBC/JDBC database host server jobs). These jobs by default use the class object QSYS/QPWFSEVER. Altering its temporary storage value would effectively limit the maximum amount of temporary storage these jobs can consume.

Here is a sample command: **CHGCLS CLS(QSYS/QPWFSEVER) MAXTMPSTG(500000)**

This sample command would limit each ODBC/JDBC connection to at most 500MB of temporary storage use.

(Continued on page 7)

(Controlling Temporary Storage Consumption—Continued from page 6)

If you don't want to implement this setting for all users hitting ODBC/JDBC interfaces, refer to the article "Route specific ODBC/JDBC users to an alternate subsystem" on page 4 of our August 09, 2005 Centerfield newsletter for the method how to create a custom subsystem and associated class object for certain users only.

For another example of the same technique refer to the following link: [http://search400.techtarget.com/tip/0,289483,sid3\\_gci782944,00.html](http://search400.techtarget.com/tip/0,289483,sid3_gci782944,00.html)

- 2) Starting with V5R4, you can control any query's temporary storage limit regardless of the interface used to execute the query by setting the STORAGE\_LIMIT setting in a global QUSRSYS/QAQQINI query control file.

Sample SQL command is: **UPDATE QUSRSYS/QAQQINI SET QQVAL = 500 WHERE QQPARM = 'STORAGE\_LIMIT'**

The nice thing about this method is that it is easily reset by updating the value back to the **\*DEFAULT** setting.

- 3) Starting with V5R4, you can use the CHGQRYA command to limit the query engine's temporary storage consumption.

Sample command is: **CHGQRYA QRYSTGLMT(500)**

This sample command would limit the current job to 500MB of temporary storage. You can of course use the **JOB** keyword to limit another user job's temporary storage, i.e.:

**CHGJOB JOB(021300/QUSER/QZDASOINIT) QRYSTGLM(500)**

- 4) Starting with V5R4 and in direct relation to the usage of options 2 & 3 (STORAGE\_LIMIT and QRYSTGLMT), you can also leverage Query Governor Exit Point (QIBM\_QQQ\_QUERY\_GOVR) to implement policy driven temporary storage management. Since you can register an exit point program for that exit point, you are free to do whatever you want in your program (audit log, email notification...). This exit point allows your program to treat different users/jobs/SQL statements in a custom manner, and then decide whether to reject the query or let it run.

There are a number of caveats with this option (i.e. exit point program is only invoked for full opens, one of the options 2 or 3 has to be in effect etc.) so take care when deciding to go with this option. Here are the details on the exit point in case you're interested in implementing it yourself:

<http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/apis/xqrygovr.htm>

### **Limiting permanent storage**

I wanted to stay away from any discussion on limiting permanent storage on the system, but keep thinking back to a prevailing misconception end users and developers have that QTEMP storage is also considered temporary storage. It is not considered temporary storage but a wayward program creating monster files can get you into the same trouble as temporary storage does, so I figured I'd briefly mention an option to control the permanent storage consumption as well. One time tested option is to limit the amount of storage a particular user profile can own.

Sample command is: **CHGUSRPRF USRPRF(ELVIS) MAXSTG(500000)**

This sample command sets the maximum amount of storage I can own to 500MB.

This setting applies to independent ASPs (iASP) as well, except that the same setting is propagated to the iASP so if I had 3 iASPs varied on, my effective limit would be 1500MB.

I am not a big fan of limiting permanent storage, but have seen a business case for limiting the storage of the developers testing new programs. Of course, the fact that it works for limiting QTEMP storage is a plus.

### **Notification of critical storage condition**

The system has a default threshold when it sends the message(s) about critical storage condition to the system operator message queue. Pretty much all of the customers I speak to have implemented some sort of elevated warning system based on that message (email, page etc.). Others have implemented custom programs that perform WRKSYSSTS polling and then send an email when certain threshold is reached. There are a number of vendors in this space now, and I'm not going to outline them all, as you're probably already aware of the options. These methods are fine in as far that they at least let you know that you have a problem. Diagnosing the problem may be challenging though.

The best tool for my money for detecting DASD spikes is the disk/HUNTER tool. It was originally developed and sold by Centerfield and has since been purchased our long-term partner S4i. If you are interested in learning more let us know.

Another tool that can give you a real time insight into the temporary storage consumption for each job is our insure/MONITOR tool. I wouldn't necessarily use it for notification purposes, but it's useful when performing root cause analysis.

Hope that helps and happy reading.





# MODERNIZED

## System i Databases: Getting the Job Done

register



WEBINAR

Wednesday, June 25

2:00 PM EDT

**Databorough.**  
producers of X-Analysis

Centerfield  
TECHNOLOGY 

**Databorough is very excited to be working with an industry leader Centerfield technology in Database management and modernization. Their experience and knowledge in the field, combined with Databorough's globally used automation tools, produces an unmatched set of solutions for problems faced by the majority of System i customers today.**

*Stuart Milligan, Director of Technical Strategy*

**We're pleased to partner with Databorough in an effort to help IBM i shops leverage their existing applications and databases without being held prisoner by them. We are particularly impressed with their highly innovative product suite. Products like X-Modernize drastically reduces the effort to modernize legacy databases and simultaneously cuts the time to re-use trusted business logic in new web-based applications. How can you lose when improving the efficiency of your programmers and your application's database?**

*Mark Holm, Chief Technology Officer*