PowerVision White Paper

Building Data-Centric n-Tier Enterprise Systems

Abstract: This white paper takes a critical look at the pros and cons of building data-centric systems and using database-based services to implement business logic, exploiting what databases do best. Data-centric enterprise systems are well suited to implement data access and business logic functionality physically in the database server using stored procedures. With Java as the leading contender for stored procedure language, this approach provides the added benefit of deploying the business logic either in application services or in data services without changing the source code.

Nitin Uplekar

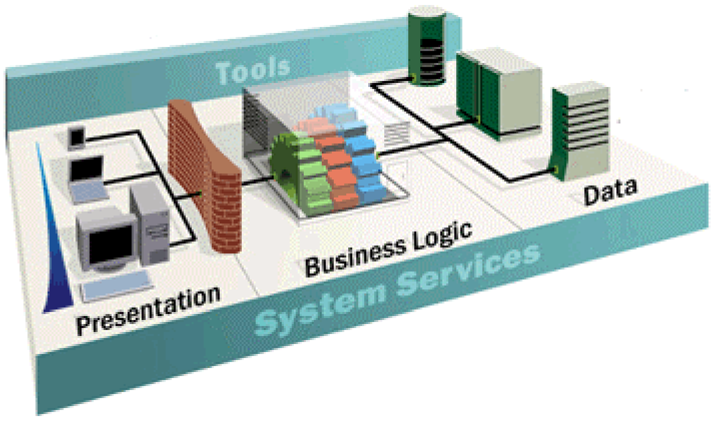
July 2001

PowerVision CORPORATION

8945 Guilford Road, Suite 125
Columbia, Maryland 21046-2391

The n-Tier Architecture Model

The three-tier architecture model, which is the fundamental framework for the logical design model, segments an application's components into three tiers of services. **These tiers do not necessarily correspond to physical locations of various computers on a network**, but rather to logical layers of the application. How the pieces of an application are distributed in a physical topology can change, depending on the system requirements.



The tiers[1] are:

- **The presentation tier, or user services layer, gives a user access to the application.** This layer presents data to the user and optionally permits data manipulation and data entry. The two main types of user interface for this layer are the thick-client application, and the Web-based application. Web-based applications now often contain most of the rich data manipulation features that thick-client applications use. This is accomplished through use of Dynamic HTML and client-side data sources and data cursors.

  Note: In a three-tiered application, the client-side application will be thinner than a client-server application because it won't contain the service components otherwise located in the middle tier. This results in less overhead for the user, but more network traffic for the system, because components are distributed among different machines.

- **The middle tier, or application services layer, consists of business and data rules.** Also referred to as *the business services tier,* the middle tier is where engineers address mission-critical business problems and achieve major productivity advantages. The components that make up this layer can exist on a server machine to assist in resource sharing. These components can be used to enforce business rules, such as business algorithms and legal or governmental regulations, and data rules, which are designed to keep the data structures consistent within either specific or multiple databases. Because these middle-tier components are not tied to a specific client, they can be used by all applications and can be moved to different locations as response time and other rules require. For example, simple edits can be placed on the client side to minimize network round-trips, or data rules can be placed in stored procedures.

- **The data tier, or data services layer, interacts with persistent data usually stored in a database or in permanent storage.** This is the actual DBMS access layer. It can be accessed through the application services layer and, on occasion, by the user services layer. This layer consists of data access components

[1] Layers and Tiers are used interchangeably in this paper.

(rather than raw DBMS connections) to aid in resource sharing and to allow clients to be configured without installing the DBMS-specific libraries and call-level interface (CLI) drivers (such as JDBC or ODBC) on each client.

During an application's life cycle, the three-tier approach provides benefits such as reusability, flexibility, manageability, maintainability, and scalability. You can share and reuse the components and services you create and distribute them across a network of computers as needed. You can divide large and complex projects into simpler and safer projects and assign them to different programmers or programming teams. You can also deploy components and services on a server to help keep up with changes, and then re-deploy them as growth of the application's user base, data, and transaction volume increases.

Typically the middle tier, the application services provider, provides services like notification, logging, security, extended metadata, user definitions, business logic, etc., and may be subdivided into one or multiple layers between the client and server, making it an n-tier application. Producing a good n-tier application often entails a series of judgments in planning and implementing the final product. When these decisions are poorly made, deployment teams can encounter time-consuming and often difficult-to-solve performance and scalability problems after the application is installed and implemented. Fortunately, many of these problems can be anticipated and prevented. This white paper deals with data-centric enterprise systems to unravel architectural misconceptions and demonstrate how to address performance and scalability problems.

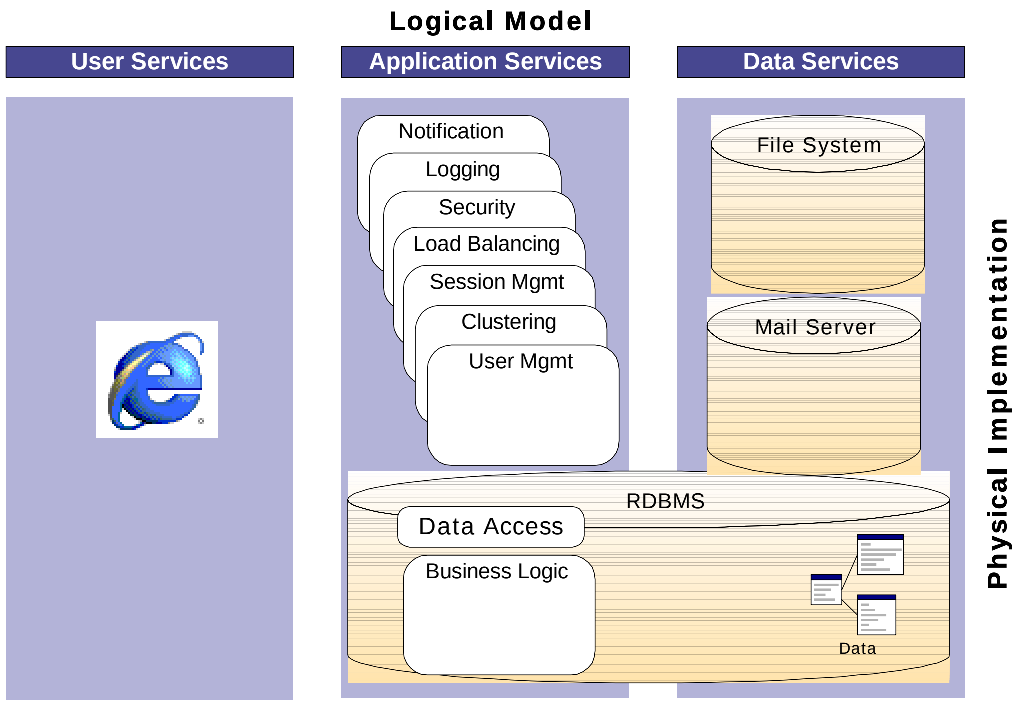
Implementing Data-Centric Tasks

The middle tier of the n-tier application is necessarily complex because of the role it plays in the overall application. One of the many tasks it performs is to receive input from the presentation layer. A middle tier is also responsible for interacting with data services to perform the business operations the application was designed to automate. Finally a middle tier returns processed information to the presentation layer.

Designers and developers often fall into the trap of including data-centered tasks with the business services in a middle tier instead of where they belong: with the data services. Rules are frequently too rigid to account for all cases, but it would be very unusual to find a justification for breaking this one. If data-intensive tasks are included in a middle tier, the system will most likely perform poorly.

For example: It would be a mistake to retrieve multiple data sets from different tables and then join, sort, or search the data in middle-tier objects. The database is designed to handle this kind of activity—moving it to the middle tier is almost certainly a bad idea. True, there may be circumstances where doing so is necessary because of the nature of the data store, but most of this should happen in the database before the dataset is returned to the middle tier.

Keep in mind that the business services are still part of the logical middle tier, and the physical implementation of the business logic may happen as part of the services offered by the relational database management system (RDBMS). However, the architecture still inherently remains multi-tier as the diagram below depicts.



All major database vendors in the market today provide robust and scalable sub-systems to implement data-centric architectures. Oracle, one of the leading database vendors in the marketplace today, offers a mature set of tools to implement the business logic using stored procedures. With stored procedures, you can implement the business logic on the server level, thereby improving application performance, scalability and security.

What are the advantages?

Performance

Stored procedures are compiled once and stored in executable form, so procedure calls are quick and efficient. Executable code is automatically cached and shared among users. This lowers memory requirements and invocation overhead.

By grouping SQL statements, a stored procedure allows them to be executed with a single call. This minimizes the use of slow networks, reduces network traffic, and improves round-trip response time. OLTP applications, in particular, benefit because result-set processing eliminates network bottlenecks.

Additionally, stored procedures enable you to take advantage of the computing resources of the server. For example: You can move computation-bound procedures from client to server, where they will execute faster. Likewise, stored functions called from SQL statements enhance performance by executing application logic within the server.

Flexibility

As Java is introduced as the language of choice for stored procedures, it added a nascent flexibility, which was not present before. Java code complying with a set of coding standards can be executed without changing a line of source code, either as a stored procedure inside the database address space or as a component within the application server address space. So it truly supports the “write once, deploy anywhere” paradigm. This provides the ultimate flexibility to the system architect. **If some organizations have concerns about implementing business logic within the database space, the same code base can be run externally under the control of the application server. This preserves the flexibility of moving the business logic into the database space in the future for ultimate system scalability and performance.**

Productivity and Ease of Use

By designing applications around a common set of stored procedures, you can avoid redundant coding and increase programmer productivity. Moreover, stored procedures let you extend the functionality of the RDBMS. For example: Stored functions called from SQL statements enhance the power of SQL.

You can use the Java integrated development environment (IDE) of your choice (e.g. WebGain VisualCafé™, JBuilder™, or JDeveloper™) to create stored procedures. Then you can deploy them on any tier of the network architecture. Moreover, they can be called by standard Java interfaces such as JDBC, CORBA, and EJB; and by programmatic interfaces in the development tools such as WebGain VisualCafé™, JBuilder™, and JDeveloper™.

This broad access to stored procedures lets you share business logic across applications. For example: A stored procedure that implements a business rule can be called from various client-side applications—all of which can share that business rule. In addition, you can leverage the server's Java facilities while continuing to write applications for your favorite programmatic interface.

Scalability

Stored procedures increase scalability by isolating application processing on the server. In addition, automatic dependency tracking for stored procedures aids the development of scalable applications.

The shared memory facilities of the Multi-Threaded Server (MTS) enable Oracle8*i* to support more than 10,000 concurrent users on a single node. For more scalability, you can use the Net8[2] Connection Manager to multiplex Net8 connections.

Maintainability

Once it is validated, you can use a stored procedure with confidence in any number of applications. If its definition changes, only the procedure is modified, not the applications that call it. This simplifies maintenance and enhancement. Also, maintaining a procedure on the server is easier than maintaining copies on different client machines.

Interoperability

Within the RDBMS, Java conforms fully to the *Java Language Specification* and furnishes all the advantages of a general-purpose, object-oriented programming language. Also, as with PL/SQL, Java provides full access to Oracle data, so any procedure written in PL/SQL can also be written in Java.

PL/SQL stored procedures complement Java stored procedures. Typically, SQL programmers who want procedural extensions favor PL/SQL, and Java programmers who want easy access to Oracle data favor Java.

The RDBMS allows a high degree of interoperability between Java and PL/SQL. Java applications can call PL/SQL stored procedures using an embedded JDBC driver; conversely, PL/SQL applications can call Java stored procedures directly.

[2] Net8 is Oracle's implementation of Transparent Network Substrate to connect a client to the server.

Security

The Oracle JServer uses Java 2 security to protect its Java virtual machine. All classes are loaded into a secure database, so they are "un-trusted". To access classes and operating system resources, a user needs the proper permissions. Likewise, all stored procedures are secured against other users (to whom you can grant the database privilege EXECUTE).

You can restrict access to Oracle data by allowing users to manipulate the data only through stored procedures that execute with their definer's privileges. For example: You can allow access to a procedure that updates a database table, but deny access to the table itself.

Any disadvantages?

Server Processing and Sizing

For data-centric systems, as you move the business logic processing from the application server to the database server, it is obvious that the processing load is being shifted, and one may need a beefier database server. Keep in mind that regardless of where the business logic is located, the SQL is still executed on the database server. This means there is an incremental load added to the database server. Our experience dictates that database servers are typically enterprise-class machines with built-in reliability, scalability and redundancy. More often than not, you will find that servers have enough capacity to handle the increased load.

Vendor Dependence

With the current activity in the technology market, customers worry about vendor dependence on a database market. The primary concerns are usually licensing costs and vendor stability. Engineers and programmers may feel that, by using database services in the form of stored procedures, one is increasing the buy-in into the vendor technology and moving away from database platform independence. This argument has merit, however as most of the major database vendors (Oracle, DB2, and Sybase—actually all except Microsoft) have now adopted Java as the primary language for stored-procedure implementation with the respective database engines. By using Java as the stored procedure language, the dependency issue is partially addressed.

Database independence carries its own cost. When a customer selects a RDBMS, like Oracle, a service-level commitment is already made in terms of operational procedures, DBA skills, backup and recovery procedures, and hardware. Even though database independence can be maintained at a software level, the cost of implementing a different database system at an operational level is significant—not including the additional licensing costs one will have to pay to acquire a different product.

In a nutshell, if you are paying premium licensing costs for a product like Oracle8*i*, you might as well exploit the full functionality of the product to get the biggest bang for your buck.

Myths Dispelled

Myth 1: Stored procedures are a DBA responsibility.

It is a common misconception that just because stored procedures reside in the database they are also administered and maintained by DBAs like other database objects, such as tables. This is far from reality. The stored procedures are executed in the database address space, however, the stored procedures are written by programmers and maintained under normal configuration control like any other source code, then administered by team leads as appropriate. Roles are assigned to the developers, and those with stored procedure maintenance roles will manage the stored procedures.

Myth 2: Stored procedures require another skill-set.

Customers often fear that the introduction of stored procedures means that they will need to maintain two different skill sets: e.g., Java for other middle-tier services, and PL/SQL for the stored procedures. With Oracle8*i*, the stored procedures can be written either in Java or PL/SQL. So if Java is the preferred skill-set in the organization, the same skill base should be used to develop and maintain the stored procedure code base. Normally programmers with expertise in SQL tend to write stored procedures in PL/SQL, while those who have Java expertise will use Java and JDBC to develop stored procedures. Regardless of where your business logic is implemented, bear in mind that the programmers must have a good grasp of SQL to code the processes in Java and JDBC.[3] Typically, SQL programmers who want procedural extensions favor PL/SQL, and Java programmers who want easy access to Oracle data favor Java. In general, there is no additional skill-set requirement for writing stored procedures.

Myth 3: *What* configuration management?

Some perceive that stored procedures are database objects and are not subjected to configuration management. Nothing can be further from reality. The same Software Configuration Management Plan (SCMP) applies to stored procedure code as it does any other Java code. The same version control system applies. The rigor of software engineering must be instituted in order to have successful projects, and all components in the system are subjected to the configuration management. A programmer follows a design specification, codes the stored procedure, tests it, checks that it's in the Version Control System, hands over the code to the test engineers who test the code,… and the cycle continues.

[3] J2EE purists will arguably state that Java programmers don't have to deal with SQL constructs when implementing container-managed persistence (CMP), and have the luxury of a rich query capability via Object Query Language (OQL).

Myth 4: Lack of debugging and development facilities.

Another common misconception regarding stored procedures is that there is a general lack of debugging and development facilities. People always fear the unknown. In the specific case of Oracle8*i*, full-fledged development environments are available for both PL/SQL and Java. With packages like DBMS_PROFILE, DBMS_TRACE, DBMS_OUTPUT, in conjunction with SQL*Plus, Oracle provides a robust development environment complete with profiling and debugging facilities. Several third-party tools are also available, including Toad by Quest Software.

As we have indicated above, if Java is your language of choice, any and all Java IDEs can be used including WebGain VisualCafé™, JBuilder™, and JDeveloper™, etc. The Java methods are developed, debugged and tested in these IDEs and they are deployed as stored procedures to be executed internally with DBMS address space, or externally in the middle tier without changing a line of source code.

Summary

Enterprise systems—typically encountered in data processing, knowledge management, and the survey and ratings industry—tend to be data-centric. Data-centric systems typically involve significant number crunching and data-set manipulation. These systems are well suited to implement the data access and business logic functionality, logically in the middle-tier, but physically as part of the database server using stored procedures. With Java as the leading contender for a common stored procedure language, this approach provides the added benefit of "code once, and deploy everywhere". The tactical design of where to deploy the business logic can thus be delayed until the deployment phase and can be easily changed at a later date. Implementing the business logic for data-centric systems as stored procedures provides enhanced performance, productivity, maintainability, and a higher level of granularity in the security and scalability.

References

- "Developing Stored Procedures in Java™", An Oracle Technical White Paper, April 1999, http://otn.oracle.com/tech/java/jsp/.
- "Developing Oracle8i Applications in PL/SQL™ and Java™", An Oracle Technical White Paper, May 1999, http://otn.oracle.com/tech/java/jsp/.
- "Building Windows DNA Applications", Microsoft Corp, http://msdn.microsoft.com/library/en-us/dndna.
- Oracle8*i* Java Stored Procedures Developer's Guide.
- "Java in the Database", David Ritter, Intelligent Enterprise, Jan 1999.
- "Architecture Decisions for Dynamic Web Applications", Gregory Lake, Microsoft Corporation, November 2000, Nile Case study, http://msdn.microsoft.com/library/en-us/dnnile/html/.
- "Options Query Tool, Case study", developed for Motorola PCS, August 1999.