

Selecting an Embedded Database for Java™ Application Development

Introduction

Today's Java software vendors are increasingly finding a market for enterprise-level applications that support a wide range of corporate initiatives. Java-enabled enterprises have shifted from early adopters, such as telecommunications and financial services, to traditional late adopters such as health care and manufacturing. According to the Gartner Group (2002), 70 percent of large-scale organizations are writing at least some applications in Java, and 48 percent are using it to create mission-critical Internet software.

Many of these off-the-shelf Java applications, middleware and tools are shipped with an integrated database – including sales force automation and employee productivity applications such as application servers, business process management tools, and Integrated Development Environments (IDEs). These databases are not used only to store end-user data, but can hold the application's operational metadata or application-specific data such as catalogs or images. Vendors may also include a default out-of-the-box database of tutorials and demos to speed customer deployment and product adoption.

Java software vendors are accustomed to creating these databases by either developing a proprietary flat-file database or licensing an enterprise system from one of the major database vendors such as Oracle®, IBM® DB2 or Microsoft® SQL Server™. However, a third option is rapidly coming to the forefront: the embedded Java database.

This paper discusses the advantages of the embedded Java database and compares them to more traditional approaches. It describes the significant development benefits achieved by leading Java software providers, including Sun, BEA and Macromedia. These include lowered costs, greater operational efficiency, shorter development cycles, ease of deployment and greater customer satisfaction.

Foremost among embedded Java database products is PointBase Embedded, a standards-based relational database. Delivering cross-platform portability, a small footprint, full Java-based security, and zero administration, PointBase Embedded offers a low-cost, high-results solution for application data management.

The Database in Java Application Development

Databases that work with a specific application provide an engine and data store dedicated solely to that application. The three most common approaches to database development utilized by Java developers are:

- 1) Proprietary flat-files systems developed in-house by the software vendor: A relatively simple approach in which data is contained in files as records with limited or no structured relationships.
- 2) Accessing an enterprise RDBMS: The application uses JDBC to access data in a dedicated schema or set of tables within a bundled enterprise RDBMS.
- 3) Embedded Java RDBMS: The application uses JDBC to access the application-specific data, but in this case the RDBMS resides on the same JVM (Java Virtual Machine) as the application. It is transparent to the user and is self-administered.

The Problems Related to Flat-Files and Proprietary RDBMS Databases

In many companies, developers still create custom database structures using flat-files. Flat-files provide certain basic benefits: They cost less initially and are easy to use without product-specific training. Non-binary flat-files can also be viewed and modified by any text editor or word processor and imported into a spreadsheet for analysis and graphing.

However, they also are the cause of several problems. Flat-files do not have the recovery and transaction integrity capabilities of an RDBMS and hence are vulnerable to corruption. They often have portability problems. They also do not provide for concurrent update access, which may limit the performance of the application. Without sophisticated indexing structures, flat-files are limited in the volume of data they can handle effectively; they provide no security and only very limited data structuring. File naming conventions must be established and respected, and reading retrieval is only performed sequentially. Management is limited to native OS file commands. For all these reasons, as the application increases in complexity, proprietary flat-file systems become more costly to modify and maintain. Eventually, they can become an increasingly high-cost item of the overall development and maintenance of the application.

The other common solution for developers is to write interfaces to an enterprise RDBMS from a major vendor such as Oracle, IBM or Microsoft, which is either bundled with the application or provided by the customer. However, this approach introduces several difficulties for the Java software vendor.

One significant issue is the complexity of installation. At the customer site, the database must be installed or upgraded separately from the application, with a DBA overseeing it to resolve compatibility conflicts with existing databases and to tune for best performance. The database process runs as a separate task, which introduces additional overhead, security concerns and complexity to the application. The likelihood of a smooth, quick and simple installation is low indeed.

Two other issues are quality assurance and support. In such an implementation, the application must be tested against and supported on multiple versions of every major RDBMS, on a variety of platforms. To do this thoroughly is complex, costly and detracts from the overall effort of making the application valuable to end-users; to not do so puts the reputation of the software vendor at risk. A database administration team is also required to tune and retune the new database to accommodate updates during the testing and production phases. Furthermore, if the application has been written to utilize Oracle, for example, it often will not work without rewriting the SQL syntax if the customer changes to IBM DB2. Throughout the application code, developers must introduce special conditional code to allow for every type of possible RDBMS.

The next problem is ongoing administration of the database. Customer parameter requirements may conflict with the needs of the software vendor's application. DBAs may make changes advantageous to the customer, only to adversely affect the vendor's application, leading to customer dissatisfaction and unnecessary support calls. When the customer upgrades the RDBMS, it may adversely affect the application. Similarly, performance tuning for the customer's applications may hurt the performance of the software vendor's application.

Finally, enterprise RDBMSs are far more expensive—the licensing agreements demanded by large database vendors are complicated, costly and time-consuming for both the software vendor and for the customer upon deployment.

Lowering the Total Cost of Ownership of "Embedded" vs. "Bundled"

A truly embedded database is integrated with the application itself, rather than being bundled with the application (see Figure 1). Embedded databases provide developers with a transparent system and a small footprint that requires little or no administration. They also have a significantly lower cost, both in terms of licensing fees and deployment. Embedded databases give vendors the simplicity of a flat-file system while providing all the reliability, flexibility and power of a fully relational database. At the same time, they avoid the problems created by utilizing a completely separate product such as Oracle, IBM DB2, Sybase® or Microsoft SQL Server.

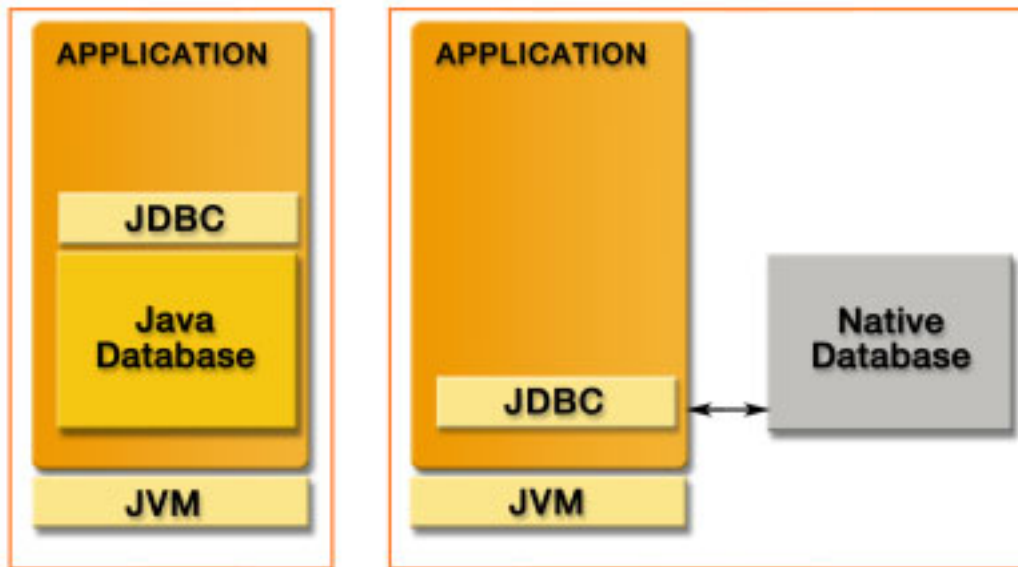


Figure 1: Java Embedded Database vs. Native "Bundled" Database

Vendors may find the language from suppliers confusing, as enterprise RDBMS systems are often described as "embedded databases." However, this terminology can be misleading. For Java developers, the best solution is a Java-based embedded database. It runs seamlessly on the same JVM as the application, giving developers the flexibility to develop, test and deploy with identical JDBC calls, SQL syntax and data types. Providing a platform-independent, multi-connection environment and data store, these databases are integrated directly with the application and require little further attention from developers.

The benefits of such a tightly integrated solution are immediately apparent within the development cycle, providing a lower total cost of ownership (TCO) for software vendors:

- Lower development costs, since developers need not deal with multiple RDBMS products or design a proprietary flat-file implementation
- Lowered costs due to reduced testing by avoiding database administration and testing on multiple platforms
- Lower installation costs for companies who must send out support staff or provide technical support to deploy applications
- No-cost administration of the database during implementation or after
- Improved customer satisfaction due to the transparency of the database
- Improved competitiveness against software vendors with a more complex installation scenario

Companies can easily save up to hundreds of thousands of dollars per year simply by not having to hire a DBA team to oversee the implementation, connectivity and tuning requirements of a bundled enterprise RDBMS. In addition, because the embedded database supports existing Java development environments, vendors do not have to invest in new IDEs or training time on new technologies. Expensive licensing fees and complex agreements are replaced by one lower cost for licensing a simple, effective embedded technology. As a result, applications have a shorter, lower-cost development cycle and may be brought to market faster.

Software vendors also achieve greater customer satisfaction and hence increased revenues from the embedded database's ease of deployment, which consists of implementing a single JAR file containing both the application and the database. Such easy and convenient installation shortens the sales cycle and improves the customer's relationship with the software vendor. It also spares them any licensing restrictions or costs imposed by major database suppliers.

For customers, cost-of-ownership studies conducted by Aberdeen prove that it is cheaper overall to use databases tied to a specific application at the workgroup level than to rely on the corporate backbone. "Until recently, there was an undercurrent in corporate IT that said, 'It costs too much to administer multiple databases. Why don't we standardize on one?' But people are moving away from that now," said Wayne Kernochan, managing vice president for platform infrastructure at the Aberdeen Group, in a recent interview with Software Developer Times (July 15, 2002).

How the Embedded Java Database is Used

Software vendors are able to leverage the embedded database in a number of different ways. It frequently serves as a store for application metadata such as application structures, classes, session states or static information such as network management and user permissions. With an embedded Java database, vendors avoid porting to multiple OS platforms and ease application upgrades, as well as reduce the inconvenience and virus issues associated with non-Java installation files – all while eliminating the data integrity issues associated with storing metadata in flat-files. Embedded Java databases can also scale to support future, larger application releases.

They are also utilized as an application-specific database, storing data related to activities within that particular application such as product catalogs or images. Embedded databases provide convenient storage for interim data collected out in the field; for example, supporting a sales force application in which the representative gathers data on a mobile unit and then synchronizes the results with the main enterprise database. Or, they can be used to support independent research or work from a laptop; for example, as a local data store for chemical research analysis performed by a scientist on the road.

Embedded Java databases are often utilized as a value-add to improve end-users' out-of-the-box or over-the-web evaluation experience with Java applications, application servers, or Integrated Development Environments (IDEs). Serving as a default database for demos, tutorials and tools, they make it easier for customers to download and install software from either a CD or over the Internet, and improve the upgrade experience as well.

What Should You Look for in an Embedded Database?

From development to final deployment at the customer site, embedded Java databases provide a lower total cost of ownership. Several factors make an important contribution to the cost-effectiveness of the embedded Java database, and they should each be made a key requirement by vendors in search of the right database technology for their applications. These include:

1) The database should be written in Java.

Allowing the database to be 100% embedded within the application, Java lets software vendors optimize the benefits of a completely transparent, implement-and-forget database, including:

- Complete portability across any platform, eliminating multiple database versions
- Flexible, cost-effective licensing
- A powerful J2EE™ security model for both database and network
- Transactional integrity via the standard Java Transaction API
- Low impact on application performance, thanks to native JDBC connectivity and running on the same JVM
- Support for major Java development environments
- Support for Java internationalization standards
- Provides the option to run the database as a server for multiple clients in a distributed environment

2) It should be robust, provide full SQL access, and be standards-based.

The database should be based on all the leading standards for databases, including J2EE certification (which includes JDBC certification), roles, triggers, stored procedures, views, etc. It should also be compliant with all major SQL features and able to synchronize data with major enterprise databases for mobile or remote applications.

3) It should have a small footprint.

Embedded databases with a small footprint allow developers to ensure minimal download time and enable easier migrations and upgrades. For customers—who are too often accustomed to enterprise databases that take up to 50MB or more of space—the embedded database should use only one megabyte or so while providing significant storage and having a low impact on downloads from either CD or the Web.

4) It should require no administration.

A self-tuning, completely integrated database does not need a DBA at any point in its usage, sparing developers concerns about constantly tuning the database. This feature shortens development cycles and brings products to market more quickly. At the same time, end-users should be able to deploy the database without concerning themselves over potential conflicts with existing databases within the environment.

5) It should be transparent to the end-user and easy to deploy.

Deployment to the customer site should be a simple, one-step process in which the application and database are downloaded together in the same executable file. The user is able to immediately make use of the application (including having instant access to tutorials and demos stored in the embedded database) without requiring the services of a DBA or systems specialist to oversee and maintain the database.

The PointBase Embedded Database

PointBase, a division of DataMirror, is the leader in the embedded Java database market. PointBase is the only company in the industry whose primary focus is on Java relational databases. Its robust, full-featured flagship product, PointBase Embedded, meets all the requirements for a complete embedded Java database.

PointBase Embedded additionally supports all standard Java IDEs, reducing development costs and increasing productivity by allowing engineers to continue working with the tools they know and prefer. No DBA is required to support developers using PointBase Embedded. With an API included for online backups, the database is administration-free. Organizations, in turn, utilize fewer resources while allowing developers to optimize the application instead of the database. PointBase Embedded maintains a small footprint (approximately 1MB) while storing up to several terabytes of data, enabling applications to scale as needed.

Increasingly, embedded databases support applications that are used across and outside the enterprise. PointBase Embedded, and its sister products – PointBase Micro and PointBase UniSync – provide the means to seamlessly support applications deployed across multiple mobile platforms, including laptops, PDAs and cell phones. PointBase Micro offers mobile users access to locally stored offline data, which can be synchronized with corporate databases on demand.

PointBase Embedded in Real-World Applications

PointBase Embedded is a proven technology that is already being utilized by a number of major software vendors.

Case Study 1: BEA's WebLogic Server Platform™ is the de facto standard for more than 13,000 customers and 2,100 independent software vendors, systems integrators, and application service providers. This leading company uses PointBase Embedded to support its out-of-the-box Web Logic Server customer evaluation experience. "Without an integrated database, we're writing toy applications," said Randy MacBlane, director of engineering at BEA. "PointBase had by far the most functionality to showcase our samples. It is a cost-effective solution that shows off the complete breadth of our products."

Case Study 2: Sun Microsystems utilizes the PointBase Embedded Java database to support customer evaluation for two of its key products: the Sun™ ONE Studio IDE (formerly Forte™ for Java) and the Sun™ ONE Application Server. The database offers a number of real-world examples to demonstrate the products' capabilities. "Sun maintains a very high level of expected quality and service for OEM products," said Dan Roberts, product manager of Sun ONE Studio. "The PointBase Embedded database has been even more successful than we expected in our customer polls – the users like it and stay with it longer than is common across the market."

Case Study 3: Macromedia, Inc. is a major vendor of software applications that are deployed to 98 percent of all web users and utilized by millions of Internet developers and designers. The company utilizes PointBase Embedded with the Server Option to present realistic examples of its JRun™ 4 server at work while connecting to four different PointBase databases. "Because we have this capability, we're able to provide a greater variety of realistic examples," said Randy Nielsen, Documentation Manager at Macromedia for JRun. "I see a special benefit in shifting the customers' focus from our printed documents to actual software samples – they aren't just reading the book, but are evaluating our products and code for themselves."

Conclusion

Increasing numbers of enterprises rely on prepackaged and off-the-shelf Java software for mission-critical business enterprise applications, middleware and tools. As software vendors build these software solutions, they are increasingly recognizing the benefits of the embedded Java database. From storing metadata and customer evaluation tools to providing an on-board database for users, Java databases such as PointBase Embedded are enabling a new class of software applications, middleware and tools that can be developed with greater efficiency, are more cost-effective, and enhance the customer experience.

Corporate Headquarters
3910 Freedom Circle, Suite 104
Santa Clara, CA 95054

Main. 650.230.7200
1.877.238.8798 (US and Canada)
information@pointbase.com
www.pointbase.com