AberdeenGroup

Profile

Lazy Software, Inc.

201 Washington Street
One Boston Place
Boston, MA 02109
Tel.: (305) 776-8055
www.lazysoft.com

Lazy Software: A New Type of Database for Faster Development and Lower Costs

Preface

Today's traditional databases are a straitjacket around development.

These databases lessen an enterprise's speed-to-market competitive advantage and increase key application administrative costs by making the programmer's life more difficult, especially in two areas:

- "Direct-to-data" programming that addresses data items directly rather than at an abstract level. "Direct-to-data" programming forces the programmer to write substantial unnecessary transactional code and makes frequently desired changes to metadata cause a "ripple effect" of crashes of undocumented applications accessing the underlying data.
- Code-to-data "impedance mismatch" in which the code is object-oriented but the data accessed is relational, causing complex coding to translate objects to relational rows and columns and vice versa.

This *Profile* describes Lazy Software's Sentences DB, Sentences DE, and LazyView. Together, these form a comprehensive database solution based on the innovative "associative" data model. This solution not only enables far faster programming of data-accessing applications, but also avoids the long-term administrative costs of "software sclerosis" caused by difficulties in upgrading legacy applications.

Executive Summary

Lazy Software's database solution includes Sentences 3.0 (which in turn includes the Sentences DB database engine and the Sentences Deployment Environment,

or Sentences DE), and the LazyView database aggregation capability. Sentences DB, written in Java, offers full database functionality using the associative model of data organization. Sentences DE, a Java applet, provides development and deployment tools for business rule, query, schema, and "profile" (Sentences DB's innovative access control and security mechanism) creation. LazyView provides an ability to view and query data across associative, relational, and other databases that matches the capabilities of Enterprise Information Integration (EII) solutions profiled in the recent Aberdeen report, *EII: The New Way to Leverage e-Information* (April 2002).

Sentences runs on Windows, Solaris, Linux, AIX, and IBM iSeries platforms. It integrates with other suppliers' databases via XML (Extensible Markup Language) and other protocols for data exchange, as well as LazyView's EII capabilities. Associative data storage can be partitioned in a highly granular way, allowing users to distribute Sentences databases across multiple platforms at the data-item level.

Aberdeen finds that Sentences is already proving its worth in the following areas:

- *Rapid development support* — Avoiding object relational mismatch and allowing transaction development at a higher level can deliver order-of-magnitude savings in new application code size and existing application upgrades.
- *Architectural flexibility* — Sentences' granular partitioning effectively supports today's "mass-deployment" architectures that distribute the embedded database of a mission-critical application across thousands of far-flung local offices.
- *Support for EII* — LazyView's ability to operate according to the associative model on relational and multimedia data gives new data-mining power to a wider class of end-users.
- *Flexible security* — Sentences DB's ability to create database partition "chapters" with end-user "profiles" allows a wider range of access control.

Lazy Software complements Sentences and LazyView with services that stress support and training on how to design and use an associative model database most effectively.

Lazy Software has staked out an area of database technology into which relational database suppliers will find it difficult to venture. Moreover, as the IT and ISV pendulum swings back toward a greater degree of new mission-critical and mass-deployment application development, the importance of Lazy Software's advantages in rapid application development can only increase. Lazy Software's Sentences and LazyView solutions are therefore worth an especially close look from IT buyers.

260 Franklin Street, Suite 1700
Boston, Massachusetts 02110-3112
Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

The Immediate Need for More Rapid Development

Aberdeen research shows that even in these times of IT cost cutting, application development and upgrading continue in many areas at a high pace. Inevitably, upkeep of existing applications requires not only problem fixing, but also adding new capabilities as organizations evolve. Proactive IT managers seek out tomorrow's competitive advantage applications today. Even cost-cutting efforts typically involve efforts to leverage existing corporate information more effectively and with more "bang for the buck," requiring new data-mining and data-integration efforts.

In all of these areas, more rapid development is a high priority. The essence of today's competitive advantage applications, such as supply chain "e-business integration" and internal customer relationship management (CRM), is speed to market, and hence, speed to program. Likewise, cost cutting where development and upgrading are inevitable must involve shortening development times — IT organizations still have not made any move to pay programmers less.

Moreover, the immediate future holds the prospect of greater need for rapid development, not less. The major new IT effort on the horizon is Web services development. As shown in the recent Aberdeen White Paper, *Web Services Development: On IT's Critical Path* (August 2002), Web services development is highly complex. To succeed, IT organizations' and ISVs' Web services development must speed development as much as possible by simplifying the task for the programmer.

The Long-Term Need to Avoid "Software Sclerosis"

The year 2000 (Y2K) problem delivered two strong messages to Fortune 1000 chief executive officers (CEOs):

1. The enterprise is *far more dependent* on legacy applications than previously imagined.
2. These applications *can be changed* en masse, using replicable processes.

In the past, the two obvious strategies that IS organizations have used to deal with legacy applications were "leaving it alone" (the "status quo" approach) and the "make-or-buy" approach. The latter approach is a radical change in existing software, and both approaches are increasingly costly ways of doing business.

Consider the status quo approach. Before the Y2K problem arrived, the typical IS organization spent little time planning to change key applications — often because it saw no low-cost way of doing so. However, what the enterprise has often failed to understand is that the net result of the status quo approach has been constantly increasing the following "inventory costs" of legacy applications:

- *Maintenance costs* — associated with outdated applications about which key information has been lost and whose software and hardware is no longer adequately supported inside the organization or by suppliers

Telephone: 617 723 7890
260 Franklin Street, Suite 1700
Fax: 617 723 7897
Boston, Massachusetts 02110-3112
www.aberdeen.com

- *Opportunity costs* — incurred as maintenance spending crowds out new application development and packaged-application acquisition spending
- *Inefficiency costs* — incurred as the failure to proactively migrate causes crisis-mode costly legacy application fixes, and periodic directives to "move to a new platform" waste time and effort on flawed or failed major software improvement projects

These costs often feed on themselves in a vicious circle, leading to software sclerosis: Choosing to maintain instead of improve an application means that the application continues to age, not only increasing maintenance, opportunity, and inefficiency costs, but also making the gap between the legacy application and today's technologies wider, making improvement more costly and access to legacy content more difficult.

"Direct-to-data" programming is a prime cause of software sclerosis. Where multiple applications use a relational database table, each must be changed when the table itself changes — because the code in each application depends on the structure of the table. Moreover, as applications age, it becomes less and less likely that IT organizations even know that the application is invoking the table — consequently, schema changes typically cause hard-to-fix crashes in one or more mission-critical applications.

Lazy Software's Associative Model

The associative model stores data items as "triples" that capture the value of the data item, its "type," and its relationship to other data items (this is a very loose description). In traditional databases (including both relational and object databases), the data items and to some extent their "types" are described in the database, whereas global types and relationships between them are to some extent handled in a separate "metadata" dictionary or as "schema declarations" in application code.

By placing type/"entity"/"object" and relationship information in the database, the associative model ensures that all application code can access the entity/relationship information rather than the code. This, in turn, means that application developers using Sentences DB and DE can avoid both the added code involved in translating object-oriented programs to relational data storage and the "software sclerosis" involved with "direct-to-data" programming. The net result for the Sentences user is not only shorter programs more rapidly developed, but also greater flexibility of the resulting code because data-access code can apply to multiple data formats of the same or associated "types." By applying LazyView, the programmer can associate and write common code not only across associative storage, but also across "views" of relational and object storage.

260 Franklin Street, Suite 1700
Boston, Massachusetts 02110-3112
Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

Associative model data is also much more easily partitioned than relational storage. In the Sentences implementation, an information store is made up of "chapters," each of which can be placed on a different machine. The information store also has a "profile" for each end-user, stating which chapters that the end-user may see. A chapter can be as small as one data item, and therefore, access control and distribution of the database can be as broad or as fine-grained as an IT organization wishes — in sharp contrast to any traditional database.

LazyView takes the ability to distribute and aggregate data to another level by extending it to non-associative information stores. Like an EII solution, LazyView collects cross-database metadata and uses it to allow end-user and programmatic transactions, such as queries across a wide variety of data stores. Because this metadata itself is stored in a LazyView database, it is exceptionally easy to use the LazyView solution to serve not only as a "database veneer," but also as a "cache" database that can store frequently accessed data from the back-end information stores.

How Lazy Software Meets the Need for Rapid Development

Lazy Software's Sentences solution speeds transactional programming by allowing programmers to refer to data at a higher, or metadata, level. Past developer experience with 4GLs (fourth-generation languages) has shown that in many applications involving a significant amount of data access, higher level data references can speed development by an order of magnitude — three weeks instead of nine months.

Moreover, developers using Sentences DE can program more rapidly because developers no longer need to write code that translates efforts to access an object class in an object-oriented programming language to efforts to access parts of a table in a relational information store — the so-called object-relational mismatch. Past studies have shown that this unnecessary code can constitute 35% to 40% of a program's total code, and it is often unusually difficult to write. Thus, it is reasonable to expect 40% time savings in many instances, as well as increasing code quality, by avoiding object relational translations.

If an application is already using Sentences DB as an embedded database, upgrading to the next application version is relatively straightforward. Because developers can program "at the metadata level," there are far fewer instances in which changing entities or relationships to accommodate new features or improve performance requires changing all applications that access the database.

How Lazy Software Solutions Meet the Need to Avoid Software Sclerosis

As noted above, "programming at the metadata level" decreases the need to change applications in order to upgrade them to the next version, making legacy application upgrading much easier. This is equally true where the application is a

legacy one that the user seeks to update in order to take advantage of new software technologies.

Moreover, the associative model makes it much easier for users to change data formats without disturbing the applications that invoke the data. Thus, users can change an entity or relationship without causing the multiple applications invoking that data to crash.

How Lazy Software Solutions Meet Database Criteria

Performance/Scalability

A surprising amount of space in relational databases is taken up by "nulls," unnecessary data items included so that each relational row will have exactly the same number of fields as any other. The associative model, while perhaps storing more "metadata" information, does allow the database designer to eliminate these null fields. The resultant space savings can have a significant impact on performance because less information stored usually correlates with fewer I/Os to access the information on disk.

A second potential scalability improvement is due to the fact that Sentences DB does not use the "update-in-place" strategy for data-item changes, which is a feature of other databases. Much of the slowdown in typical database performance as the transaction rate increases is caused by the typical database locking the data item being updated until the changed item is copied back to disk — preventing all other operations on the data item until the lock is released. By contrast, Sentences DB affects all updates (and deletions) by adding triples to the database. Although this kind of data preservation could potentially add space to the database and hence slow performance, the tried-and-true methods of archiving and reorganizing should avoid any such slowdown.

A third potential performance improvement comes from the fact that relationships between data items are stored in the data itself. For large-scale or complex querying, this means that rather than "join" different sets of data to determine if data meets a criterion, a query can tell by looking at the data — saving significant numbers of I/Os per item.

Manageability

There is, of course, no doubt that there is a learning curve involved in making full use of the associative model for administrators used to the arcana of relational database administration. However, Sentences compensates for this initial awkwardness by decreasing the administrator's job thereafter.

In particular, as noted above, Sentences requires much less frequent and easier changes to the metadata when new application functionality is desired. Moreover,

Sentences' chapter/profile access control gives the administrator a much finer degree of control and greater flexibility over end-user access to the database.

Flexibility
As noted above, Sentences' chapter/profile approach and storage of old data gives much more flexible access to particular end-users. The highly granular division into chapters also gives IT organizations a wide variety of options in partitioning and replicating portions of the information store across platforms and geographies. This highly distributed architecture is particularly suited for mass-deployment situations, in which hundreds or thousands of database copies serve as embedded databases for a key application that runs at the local office level, with optional data fed back to a central store.

Programmer Productivity
As noted above, Sentences and LazyView can be particularly effective in supporting rapid development of new applications or rapid, easy upgrading of existing ones.

These solutions are also appropriate for Web services development. Sentences DB is written in Java, Sentences DE produces Java applets, and Sentences supports XML — all key elements of today's Web services development architecture.

Where Lazy Software's Solutions Can Be Most Effective
Lazy Software's solutions can be especially effective where users aim to rapidly develop new applications, particularly those involving workgroup and/or geographically distributed database architectures (the "mass-deployment architectures" cited above), both in-house and ISV-supplied architectures. In these cases, Sentences' ability to avoid object relational mismatch and direct-to-data programming should allow exceptionally speedy development, whereas the chapter/profile approach should provide exceptionally flexible database distribution.

Lazy Software's solutions are also apposite where IT organizations seek to create applications that will require frequent upgrades — in some ways, a description of all new applications! In these situations, Sentences encourages coding practices that are much less likely to involve further upgrades when data relationships must be changed or new data added.

Where users are providing EII capabilities to an enterprise portal or Business Intelligence (BI) solution, LazyView can be particularly useful. LazyView's ability to combine EII's capabilities with exceptionally "rich" metadata and caching capabilities makes it a highly flexible and potentially high-performance tool for these purposes.

260 Franklin Street, Suite 1700
Boston, Massachusetts 02110-3112
Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

Aberdeen Conclusions

Aberdeen finds that Lazy Software's Sentences and LazyView solutions are able to combine immediate utility in rapid application development with solid long-term value in mass-deployment and EII applications. The associative model that underlies these solutions ensures that other databases will find it difficult to match these capabilities over the next few years.

Of course, as a specialized database, Lazy Software is not in direct competition with enterprise databases. Rather, it should be viewed as a new and powerful tool for key upcoming projects, with strong long-term potential as a solid base for mass-deployment applications. In those situations, Lazy Software's solutions are worth immediate and careful attention from IT buyers.

Aberdeen Group, Inc.
260 Franklin Street, Suite 1700
Boston, Massachusetts
02110-3112
USA

Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

Aberdeen Group is a computer and communications research and consulting organization closely monitoring enterprise-user needs, technological changes and market developments.

Based on a comprehensive analytical framework, Aberdeen provides fresh insights into the future of computing and networking and the implications for users and the industry.

Aberdeen Group performs projects for a select group of domestic and international clients requiring strategic and tactical advice and hard answers on how to manage computer and communications technology. This document is the result of research performed by Aberdeen Group. It was underwritten by Lazy Software, Inc. Aberdeen Group believes its findings are objective and represent the best analysis available at the time of publication.