DATA BASED DOWNSIZING: A Case Study that Exposes the Pitfalls!

DATA BASED ADVISOR®

For Developing Desktop Database Applications

March 1992 $3.95 Canada $4.95

SPEED UP THAT APP!

Tuning Tips, Tactics, and Trade-offs for:

- AREV
- Clarion
- Clipper
- DataEase
- dBASE IV
- EB
- FoxPro
- Interbase
- Oracle
- Paradox
- R:BASE
- SQL Server
- SQLBase
- SQLWindows

Plans for growing your VAR business

Nantucket's RDDs: why they're so late...

(see page 134)

03

0 74470 12914 1

Vol. 10, No. 3 March 1992

THE EXPERT SERIES

TEST DRIVES

Page 58

DEPARTMENTS

CLIENT/SERVER ADVISOR

CLIENT/SERVER ADVISOR

Looking Beneath the Surface

With the plethora of front ends available, you have to look past superficial functionality when making your selection ...

By David McGoveran

There are probably over 100 front ends on the market, but few suited for serious client/server computing and relational databases. Don't get me wrong; the needs of casual users, read-only applications, or applications that don't manipulate shared data are well-served by most existing front-end tools. These are generally easy to use, and their functionality steadily improved. But what about the concerns of corporate developers? I've spent a number of years evaluating relational database product functionality and auditing vendors' design and development practices. This has led me to the unhappy conclusion that most tools abuse the relational model (defeating its tremendous potential) and treat server data as though it were under local control (creating a variety of maintenance, performance, and development difficulties). At best, the risk of having to *redevelop* applications is too high to permit tool standardization. At worst, the vendors engage in such poor design and development practices that their ability to support the product is jeopardized.

A closer look

Users selecting a front-end tool for production applications need to perform a serious risk assessment before training, design, and development resources are dedicated to a product. Evaluating superficial functionality (i.e., functionality as understood by the end users) has been examined extensively in professional journals, but isn't enough to ensure the appropriate use of corporate resources. Risk assessment requires looking at functionality "under the hood," and evaluating the vendor in various ways.

What factors should influence risk assessment of front-end tools for production client/server applications? Here are some questions to ask:

- What assumptions does the tool's architecture make about SQL support? For example, it may not handle subqueries under the assumption they're "too complex." It may assume that all joins are based on an equality condition (ignoring other relationships such as greater than or inequality), or that all search conditions should be combined by conjunctions (i.e. by AND, not OR). Production applications require access to the entire SQL dialect supported by the server. SQL is already an impoverished language without tool vendors reducing it even more. If the tool can't generate appropriate and efficient SQL, a mechanism whereby the generated SQL can be augmented or fine tuned is required.
- What assumptions does the tool make about transaction management? Is each row update propagated to the database and committed immediately, or can the user send multiple row updates as a single transaction? The problem is a difficult one. Designers must create an interface in which the boundaries of a logical unit of work are intuitive and obvious, and correlate SQL transactions with these. In my experience, tool design-

Jill Rappaport

ers often have difficulty understanding that any data (from the database) available to the user at the commit point may have influenced an update, and therefore, should be within the transaction boundary. There are exceptions, but in a production environment, tool vendors must protect the integrity of the database. To do this, vendors must understand the transaction model (and concurrency model) used by the database server and design the tool to use that model appropriately.

- How does the tool communicate with the database, via dynamic embedded SQL or an API? Current implementations of embedded SQL have many limitations, but can often be circumvented if the tool uses an API. Arguments in support of embedded SQL "standards" carry no weight; existing standards don't yield portable code, provide equivalent behavior (consider error processing), or support SQL extensions specific to a particular database. Production applications need all the functionality they can get. For example, Sybase's APT Workbench provides both an API and an integrated SQL dialect called APT-SQL; the result is a powerful tool for database manipulation.

- How does the tool determine the structure of the database? If it queries the system catalog only at the beginning of a session, efficiency can increase at the expense of tool/database synchronization problems. For example, the structure may be changed during a session by another user, leading to hard-to-explain errors. If the tool queries the system catalog prior to each query (some tools do), synchronization problems are less likely, but performance is degraded.

- How well does the tool handle many-to-many relationships, or does the vendor assume that anything beyond one-to-many (master-detail) relationships is too complex (or worse, that the tool need handle only one table at a time)? Similarly, can master-detail-subdetail relationships be handled? Even if these are permitted visually, they're generally supported via an inefficient and non-relational approach. I have often been amazed to hear vendors discuss the unsophisticated requirements of users! Tools like JAM/dbi provide support for complex relationships and leave it to the developer to decide what the user does and does not need.

- How does the tool handle results? Tools that cache results may be fine for analysis and reporting functions, although limitations on cache management may cause problems with large result sets. If the results are used in a way that affects subsequent updates, greater care is needed to ensure synchronization between the front-end tool and the server. If a tool supports SQL updatable cursors, users should demand an explanation from the tool vendor as to how partial results are cached (in the tool or in the database), and how synchronization is maintained. I've found that tool designers rarely understand the complexities of this problem. While OS/2 Data Manager may not hide the difficulties of distributed database support, IBM has expended much effort to provide appropriate options to the system administrator and to protect corporate data. Remote Data Services supports optional front end caching through a block fetch capability.

- What assumptions are made about concurrency control? All too often, tools default to an extreme. They use either the lowest level of isolation permissible to improve concurrency, or they assume the data is owned completely by a single user and can be locked at the expense of concurrency. A particularly insidious implementation tries to simulate "optimistic concurrency" by deferring all locking until a row is actually updated. At that time, the row is checked to see if another user has changed it since it was retrieved from the database and displayed to the user. A warning is returned if the row has changed. While this technique sounds good, it ignores that fact that data from more than the one row may have influenced the user's changes, and it allows those influencing rows to change without returning a warning. This is complicated by the fact that different database servers use different concurrency control models. I don't consider any products to be adequate on these points; a few, like Oracle's SQL*Forms do provide appropriate commit and concurrency control when used with an Oracle database.

- How complete is multi-database support? For example, can the tool connect to multiple database servers? Is the connection limited to one database at a time, or can they be "simultaneous?" Are database servers from multiple vendors simultaneou supported? The need to access mu ple databases is a common requi ment in corporate environments. P viding such support in a front-e tool requires an understanding of tabase servers, error recovery, tra action management, concurren and other issues. Understanding h to manipulate the data across mul ple database servers is *not* enough create a stable environment.

- Does the product support distribut transactions? If a tool provides acce to multiple data sources, users shou be concerned about using the tool f updates. Few tool vendors unde stand the problems of distribut transaction management and ev fewer treat it seriously; its impact both database integrity and t meaningfulness of results is poor understood. Additional questions: two-phase commit supported and, so, how are transaction commit, rol back, and recovery coordinate across data sources? How is distril uted deadlock handled? What optim zations are performed (for exampl to prevent a distributed join from b ing processed in the tool as a simpl sort merge)? There are circumstance where these issues can be justifiabl ignored, but users shouldn't readil assume those circumstances are me A careful analysis is required.

- How experienced is the vendor's de sign and development team? Fe front-end designers have ever devel oped a production class client/serve application. The second-hand knowl edge accumulated from the vendor customers isn't enough. Designer and developers need to internaliz relational and client/server concepts so that appropriate design decision are second nature. For example, th tool should "understand" the con cepts of primary and foreign keys an support them if possible. Similarly the tool should be architected to mini mize control coupling between client and server during a transaction (this is fundamental to good client/server application design). Beware vendors who respond to functionality questions with the "Why would you want to do that?" syndrome.

- How formal is the vendor's design, development, and quality control? Poor coding practices lead to unexpected tool behavior. Poor documentation at any phase of the life cycle means that the user is less likely to obtain new functionality or an impor-

tant bug fix quickly. A less formal quality control program means that users have to debug the tool. The number of PC vendors who abhor formality is appalling. Nonetheless, some vendors practice good software engineering. For example, I've been impressed by Microrim's Vanguard design and development team.

- Finally, does the tool have a "clean" architecture? Is each facility designed as a "service" with a uniform interface so it can be modified as needed? If the vendor has created separate services for network interfaces, transaction management, results and cache management, the database server interface, the user interface, etc., the tool can be adapted to the corporate user's changing requirements without introducing changes where they aren't desired. This is particularly important when new database servers (including versions) or new network protocols need to be supported.

I haven't addressed all the issues involved in selecting a front-end tool. I've ignored the standard functionality issues that appear so frequently in other magazine articles. I don't mean to imply they aren't important; test driving the product and reading the manuals are just as necessary in tool evaluations, whether the user is part of a large corporation or a small business. However, it's the "proprietary information" of the product that determines whether the tool will stand the test of time and provide reliability and extensibility in the face of corporate production demands. The issues I've raised aren't meant to demean front-end tool vendors or their design and development teams. Rather, the intent is to point out the complexity of the problems and the relative immaturity of the client/server and relational database industry. With careful tool selection and application design, great success is possible with the technology already at hand.

David McGoveran is president of Alternative Technologies, a Santa Cruz, CA consulting firm specializing in solving difficult relational and client/server problems for over a decade. The material in this article was adapted from the author's book, *An Advanced Guide to Client/Server Applications*, in preparation. He can be contacted by telephone at (408) 425-1859. ❐