

**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 APPLI

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)



THE EXPERT SERIES

PARADOX

Options for Maintaining Indexes

More options for systems administrators—adding and deleting secondary indexes.

By Tim Colling 142

CLIPPER

Shady Characters

Add shadows to windows and boxes without writing C code.

By Mike Schinkel 144

WINDOWS

COMMon Problems

Trouble with Windows communications? Here are some solutions.

By Kenn Nesbitt 146

FOXPRO

Learning By Example

Use FoxPro's power tools to create a simple, scrolling list—and learn how the language works.

By Y. Alan Griver 148

dBASE IV

Using UDFs to Generate Reports

In much the same way you use UDFs to enhance editing features, use them to generate reports.

By Michael Liczbanski and Susan Perschke 150

TEST DRIVES

SOFTWARE UPDATE: Dr. Switch Adds Bells and Whistles

When you want to make your Xbase application a TSR, Dr. Switch-ASE can do. By Jeffrey Bersin

18

SOFTWARE UPDATE: MemoPlus Takes Care of Pesky Memos

Fix, pack, or modify your memo fields with this handy set of utilities.

By James Powell 20

FIELD REPORT: Clipper Developers Pack First Russian DevCon

An interesting perspective of the first Nantucket DevCon held in Russia.

By Michail Mikhailov 22

SOFTWARE PREVIEW: What's new in dBASE IV 1.5

A quick rundown of the new commands and functions.

By Michael Liczbanski and Susan Perschke 24



Page 58

DEPARTMENTS

FEEDBACK

Letters from our readers. 16

SOFTWATCH

26

DB CONNECTIONS

Keeping Up With the Pack

Stay on top of what's current with CIS and the DBADVISOR Forum.

By Bill House 158

EXPRESS LANE

161

USER GROUP

167

ADVERTISING INDEX

169

CALENDAR OF EVENTS

168

CLIENT/SERVER ADVISOR

FRONT END TOOLS

Looking Beneath the Surface

With so many front ends on the market, it's hard to choose. Here's a list of considerations that go beyond the interfaces...

By David McGovern 58

CORPORATE COMPUTING

The Time It Takes

Don't get caught up in the hype; allow extra time in laying the foundations for future client/server projects.

By Bob Zurek 62

MEMO TO MANAGEMENT

Client/Server Applications as Corporate Assets

What makes an application valuable? Here are some important application design considerations.

By Tipton Cole 64

WINDOWS FRONT ENDS

Ease Windows Development with SQLWindows

It's no secret that Windows is easy on users and hard on developers. Take a look at what Gupta's SQLWindows has done to improve developers' outlooks.

By Ramesh Balwani 66

SERVER TUNING TIPS

Emerald Bay

By John F. Hey 70

Interbase

By Michael Masterson 74

Oracle

By David McGovern 76

SQLBase

By Douglas Goddard 78

Microsoft SQL Server

By Kelly Gillespie 80

DATA BASED DOWNSIZING

The Ups and Downs of Downsizing

Here's how one development team used C to downsize a "corporate-critical" application from a mainframe to PCs.

By James Foye 82

How to Specify a SQL Server Sort Order

88

SYSTEMS INTEGRATION

Remote Client/Server Computing

By Kelly Gillespie 90

SERVER SCENE

The latest database server news. 93

PACKETS

Late-breaking client/server news. 94

CLIENT/SERVER ADVISOR



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 gener-

■ Looking Beneath the Surface By David McGoveran	58
■ Corporate Computing By Bob Zurek	62
■ Memo To Management By Tipton Cole	64
■ Windows Front Ends: SQLWindows By Ramez Balwani	66
■ Server Tuning Tips	70
■ The Ups and Downs of Downsizing By James Foye	82
■ Systems Integration By Kelly Gillespie	90
■ Server Scene	93
■ Products	94

ally 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.

Looking Beneath the Surface

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

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 conjunction (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-

CLIENT/SERVER ADVISOR FRONT-END TOOLS

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 simultaneously supported? The need to access multiple databases is a common requirement in corporate environments. Providing such support in a front-end tool requires an understanding of database servers, error recovery, transaction management, concurrency and other issues. Understanding how to manipulate the data across multiple database servers is *not* enough to create a stable environment.

- Does the product support distributed transactions? If a tool provides access to multiple data sources, users should be concerned about using the tool for updates. Few tool vendors understand the problems of distributed transaction management and even fewer treat it seriously; its impact on both database integrity and the meaningfulness of results is poorly understood. Additional questions: two-phase commit supported and, so, how are transaction commit, rollback, and recovery coordinated across data sources? How is distributed deadlock handled? What optimizations are performed (for example, to prevent a distributed join from being processed in the tool as a simple sort merge)? There are circumstances where these issues can be justifiably ignored, but users shouldn't readily assume those circumstances are met. A careful analysis is required.
- How experienced is the vendor's design and development team? Few front-end designers have ever developed a production class client/server application. The second-hand knowledge accumulated from the vendor's customers isn't enough. Designers and developers need to internalize relational and client/server concepts so that appropriate design decisions are second nature. For example, the tool should "understand" the concepts of primary and foreign keys and support them if possible. Similarly, the tool should be architected to minimize 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-

CLIENT/SERVER ADVISOR FRONT-END TOOLS

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 McGovern 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. □

FORGET THE REST GET THE BEST in Clipper® productivity

GRUMPFISH LIBRARY . . .

Grumpfish Library makes your applications more powerful and more appealing, while cutting down on your development time. You will not have to waste hours learning a new language to use Grumpfish Library — even new Clipper developers can integrate Grumpfish modules into their programs within minutes of breaking the shrink-wrap.

Grumpfish Library is written 99.9% in Clipper, and all 5.01 and Summer '87 source code is included (nearly 900K worth). Not only can you modify the source code to suit your specific needs, but it will inspire you to produce more efficient and powerful code of your own!

Grumpfish Library is not merely 5.01 compatible — it has been completely rewritten from the ground up to take advantage of all the powerful Clipper 5.01 features.

Grumpfish Library Features

- Pop-up desktop utilities: spreadsheet (NEW), calculator, calendar, appointment, tracker, notepad, stopwatch
- Numerous extensions to the GET command (CALCULATOR, LIST, PROPER, etc.)
- Generic database browser with automated record layout and edit view. QBE routines. Includes a built-in screen painter that lets you design and generate Clipper code for your data entry screens in seconds!
- Nearly 100 functions to make your Clipper development life easier
- Dynamically overlayable by the current crop of dynamic overlay linkers
- 30 days free voice support, one year unlimited free BBS support
- Handsome printed documentation, Norton Guides reference database
- No hassle, unlimited royalty-free integration license

GRUMPFISH MENU . . .

Slash your development time with Grumpfish Menu! Create working prototypes in minutes instead of days. Handle change orders in seconds instead of weeks! If you can create a text outline file, you can create a gorgeous easy-to-use front end with Grumpfish Menu — it's that simple!

There is no need to learn a template language — just store your menu structure in a text outline file, and Grumpfish Menu instantly generates optimized, ready-to-compile Clipper source code for either Summer '87 or 5.01! Now you can change your menu structure as much as necessary without having to worry about screen housekeeping or case logic.

Grumpfish Menu is shipped with two versions of the menu generator and linkable libraries, one for Summer '87 and one for Clipper 5.01. The supporting functions for the menu system are written 96% in Clipper (Assembler is used only for transparent shadowing and certain string handling DOS functions).

Grumpfish Menu Features

- Seven menu styles including pull-down, cascading 1-2-3, and boxed 1-2-3
- Hot keys on any menu item (NEW)
- Security levels on any menu item (NEW)
- Mobile and resizable menus (NEW)

- Built-in tutorial system
- Unprecedented aesthetic control — change menu attributes from within your application without recompiling or relinking!
- User-defined configuration files can be loaded at run-time on the command-line — each user can have their own interface!
- 30+ different configuration options
- Develop quick prototypes by toggling one configuration option. Grumpfish Menu will ensure that calls to non-existent functions will be handled gracefully, rather than crashing your program.
- Embed source code anywhere within your menu structure
- Define your own custom entrance and exit routines
- Allow users to toggle menu options or log multiple options by adding one line to your menu outline file
- Seamless support for Grumpfish Library desktop utilities and Dr. Switch-ASE™
- 100% compatible with the current crop of dynamic overlay linkers
- 30 days free voice support, one year unlimited free BBS support
- Handsome printed documentation

Available through

**PROGRAMMER'S
Warehouse**

7819 E. Greenway Rd., Ste B
Scottsdale, AZ 85260
1-800-323-1809 FAX (602) 441-0679

1-800-323-1809
Call now for
a free
demo
disk

GRUMPFISH, INC.

2450 Lancaster Dr. NE, Suite 206, Salem, OR 97305
Tel: (503) 588-1815, Fax: (503) 588-1980 BBS: (503) 588-7572, CompuServe: 70673355
Grumpfish — Your Guide to Clipper Education and Productivity