

# CLARIFYING CLIENT-SERVER

*Client-server architecture has advantages, but not necessarily distributed database support.*

by David McGoveran and Colin J. White

*Editor's note: Much of DBMS magazine's treatment of client-server architecture focuses on LAN applications in which a single database server running on an Intel-architecture machine supports DOS and/or OS/2 clients, or a single database server running on a Sparc-architecture machine supports UNIX clients.*

A client-server architecture on LANs can support transaction processing applications. Such architecture is a less expensive alternative to minicomputer and mainframe platforms, with the added bonus of providing data processing operators with the PC's responsive user interface. Database server software brings the power of the mainframe DBMS down to the LAN, offering security, concurrency control, and transaction logging that go far beyond what is available from DBMSs with a PC heritage that operate in a file server mode.

A client-server architecture on LAN platforms also supports a new style of end-user computing. The performance and integrity provided by database server software allows a network of end users to share an organization's data via their favorite applications. Some can run spreadsheets, such as Microsoft Excel with its Q&E link, Wingz with its link to Informix, or Lotus 1-2-3 with its Datalens link. Others can run their customary PC DBMSs via new database

Colin J. White is President of Database Associates in Morgan Hill, Calif., and Editor/Publisher of InfoDB and Database Review. David McGoveran is president of Alternative Technologies in Santa Cruz, Calif.

server links (available now for DataEase, Paradox, and Advanced Revelation to name a few, with dBASE IV and others soon to follow). Still others can run new executive information tools for ad hoc query and data analysis such as Channel Computing's Forest and Trees. Beginning with the December issue of DBMS, we will focus on this deluge of client-server tools with a new column called "On the Front End."

DBMS magazine's vision of client-server architecture as a better alternative to file server architecture on LAN platforms is clear. Even so, we are aware that the world is bigger than the extent of the LAN, and we are aware that the picture of client-server in that larger world is not so clear. The term server is applied variously to machines that provide file services, to machines that provide database services, and (correctly) to the logical process that provides database services. The vision becomes even blurrier when mainframe and minicomputer platforms replace LANs as the focus of the picture. What does client-server technology mean to the MIS shop more used to thinking in terms of CASE tools and COBOL than in terms of Clear+ and Paradox? What if the main, "mission-critical" manipulation of the data is to remain on the larger platform and only extracts or snapshots of the data are to be made available to end-user workstations?

There's little question that client-server technology is an opportunity for MIS to provide cheaper yet higher-quality access to data without letting that data out from MIS's protective control. Client-server architecture can offload end-user computing, application development, and, yes, even trans-

action processing from large machines to the cheap MIPS offered by PCs and workstations. Client-server is MIS's ace in the hole to stave off ever-degrading response times and expensive processor upgrades.

Unfortunately, the clarity of this promise is often muddled by the intermingling of distributed database concepts into discussions of client-server architecture as if they were one and the same. They are not, although some marketing hype might lead you to such a conclusion. Client-server databases are here today, while the first pieces of true distributed databases are just falling into place.

To help clarify the picture, DBMS Advisory Board member Colin White and his colleague David McGoveran herewith tackle the task of rigorously defining client-server architecture. The concepts covered in this article can be applied to any client-server product on any hardware platform, but are particularly appropriate for MIS shops that want to understand the implications of distributed access to data stored on minis and mainframes.

The data that most large organization applications access is stored in central databases. The use of mini- and mainframe computers for this centralized application processing is becoming increasingly more expensive, especially when compared with the price/performance ratio of microcomputer systems. For this reason, ways of off-loading minicomputers and mainframes to cheaper microcomputer or workstation solutions is of increasing interest. End-user computing, applications development, and corporate processing are all candidates for off-loading and downsizing.

Even if these types of processing are moved to a workstation, there is still a need for users to access host data. A client-server architecture is one way to resolve the issue of how applications and tools can get easy access to this data.

## Terminology

Like many other data processing technologies, client-server is littered with confusing terms, such as distributed processing, distributed database, cooperative processing, and peer-to-peer, to name but a few. In order to understand and distinguish among these, we'll first define a few of the key terms.

**Server.** A server is a logical process that provides services to requesting processes. In general, it does not send results to the requester until the requesting process tells it to do so. It is up to the server to manage synchronization of services and communications once a request has been initiated. The use of the term server in this article should not be con-

fused with a piece of hardware, i.e. a special-purpose processor dedicated to running server software. There are many kinds of servers, including network, file, terminal, and database servers. A database server is the logical process responsible for processing database requests. In this article, we will restrict the discussion to database servers.

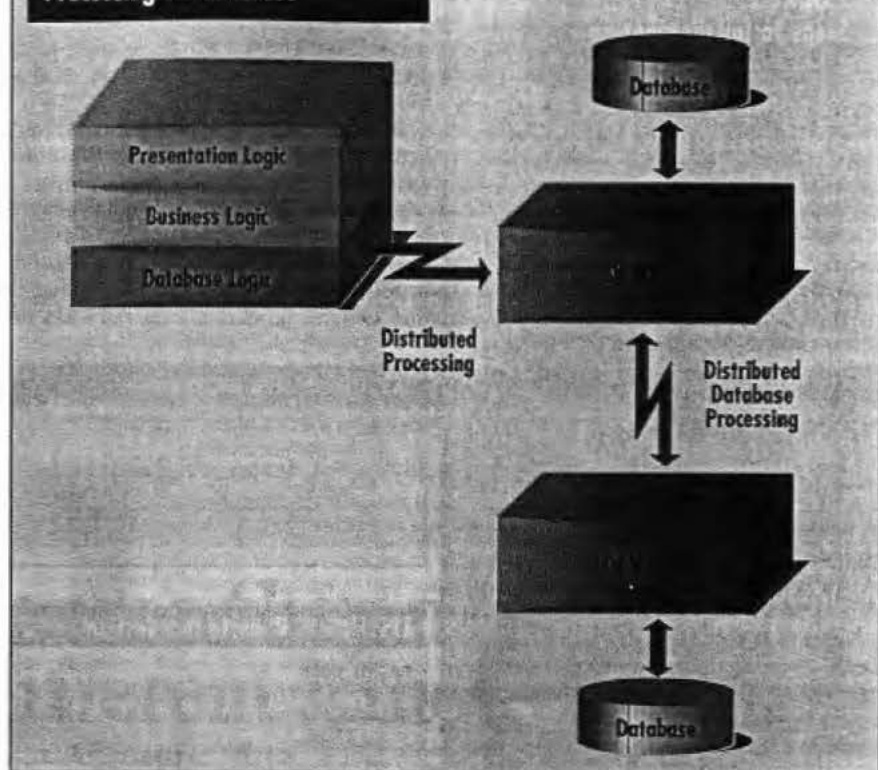
**Client.** Processes that request services from a server are called the clients of the server. As above, the term client should not be confused with hardware, i.e. processors connected to hardware "servers." One characteristic that distinguishes a client from its server is that the client may initiate a communications transaction (not necessarily a database transaction) with the server, but the server never initiates a communications transaction with the client. It is the task of the client to initiate communications, request specific services, acknowledge services completion notification, and accept results from its server. While the client may request either synchronous or asynchronous notification of service completion, it does not manage synchronization of services and communications. In a client-server architecture, many clients may "share" a single server.

**Client-server communications.** Communications between client and server in a particular installation can involve a variety of mechanisms: LAN, WAN, or operating system task-to-task communications services via mailboxes, shared memory, named pipes, and so on. However, a client-server architecture should be independent of these methods and the physical connection between them. A client-server architecture supports transparent reconfiguration or even replacement of the client-server communications interface so that applications and database processing need not be altered. In particular, note that the client and the server need not be on physically distinct processors or nodes. If the user decides initially to locate a client on the same physical machine as its server and use shared memory for communications, then later to locate them on geographically separated machines and use a satellite for communications, the architecture should support the change transparently.

### Distributed Architectures

The term client-server is frequently associated with and often used almost interchangeably with "distributed." This is due to the fact that a client-server architecture provides good support for distribution. Indeed, the kinds of distribution a client-server architecture supports can be used to classify client-server database applications. For this reason, we now take a look at the different kinds of distributed application, and define the types of distributed capability required by each.

**FIGURE 1**  
Processing vs. database



**Distributed processing and distributed database are not synonymous. Distributed processing that involved a database application involves doing the presentation, business, and database logic processing on one computer and the database processing on another. Distributed database, in contrast, involves taking a database and spreading the data across a set of computers.**

We assume throughout that the reader has a basic understanding of distributed database technology and its advantages [Editor's note: see Herb Edelstein's article in the September 1990 issue of *DBMS*].

The main objective of a client-server architecture is to allow client applications to access server managed data. The server could be running on a remote computer — for example, a mainframe across the country or a 486-based machine across the LAN. For this reason, client-server applications are frequently associated with the term distributed processing. A client that is physically separated from the server engages in distributed processing, but distributed *database* processing is not necessary for client-server computing, nor is physical separation a requirement for client-server processing.

As implied above, care must be taken to differentiate distributed processing support and its forms from distributed database support and its variations. Distributed processing involves taking some processing and spreading it across a set of computing resources. In a database application, distributed processing could involve doing the presentation, business and database logic processing on one computer (typically an intelligent workstation), and doing the database processing on another. Distributed database, on

the other hand, involves taking a database and spreading the data across a set of computing resources (see Figure 1).

The splitting of the data in a distributed database application may be done by storing different database tables on different computers, or even by storing different parts or fragments of an individual table on different computers. No matter how the division is done, it must be transparent to the application, i.e., the application (or the user, for that matter) should not be aware that the data is distributed if the database is to be truly distributed.

In the following sections, we examine the features that distinguish various types of distributed database processing, methods of data distribution, and the requirements for distributed architectures in more detail. The reader should be aware that, although discussed in terms of client-server architectures, these types of distributed data processing can be more broadly applied.

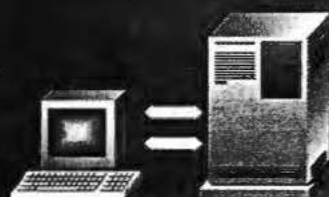
### Distributed Data Processing

The objective of the distributed systems we are about to describe is to give application users access to both local and remote data. There are several types of request an application on a client can send to a (remote) database server for

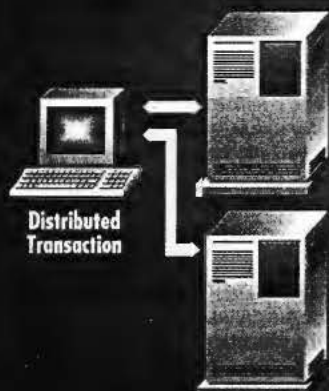
**FIGURE 2**  
Steps to full distribution



Remote Request



Remote Transaction



Distributed Transaction



Distributed Request

processing (see Figure 2):

**Remote request.** A remote request capability allows a single request (e.g. a single SQL statement) to be sent to a single server for processing, as shown in Listing 1.

**Remote transaction.** A remote transaction capability allows a transaction consisting of multiple requests to be processed at a single (perhaps remote) server, as shown in Listing 2.

**Distributed transaction.** Distributed transaction capability allows a transaction consisting of multiple requests to

be processed by multiple servers (local or remote). Each request can be processed only by a single server, but different requests within the same transaction can be processed by different servers, as shown in Listing 3.

**Distributed request.** A distributed request capability allows a transaction consisting of multiple requests to be processed by a distributed database server (local or remote). Each request can be processed by multiple physical servers, but this is transparent to the client. Distributed request processing allows, for

**LISTING 1. Remote request**

```
SELECT A.*
FROM
  SERVER1.DATABASE1.TABLE1 A, SERVER1.DATABASE1.TABLE2 B
WHERE
  A.COLUMN1 = B.COLUMN1
```

**LISTING 2. Remote transaction**

```
BEGIN WORK
SELECT A.*
FROM
  SERVER1.DATABASE1.TABLE1 A, SERVER1.DATABASE1.TABLE2 B
WHERE
  A.COLUMN1 = B.COLUMN1
UPDATE SERVER1.DATABASE1.TABLE1
SET COLUMN1 = "NEWVALUE"
COMMIT WORK
```

**LISTING 3. Distributed transaction**

```
BEGIN WORK
SELECT A.*
FROM
  SERVER1.DATABASE1.TABLE1 A, SERVER1.DATABASE1.TABLE2 B
WHERE
  A.COLUMN1 = B.COLUMN1
UPDATE SERVER2.DATABASE1.TABLE1
SET COLUMN1 = "NEWVALUE"
COMMIT WORK
```

**LISTING 4. Distributed request**

```
BEGIN WORK
(SELECT A.*, B.*
FROM
  SERVER1.DATABASE1.TABLE1 A, SERVER1.DATABASE1.TABLE2 B
WHERE
  A.COLUMN1 = B.COLUMN1)
UNION
(SELECT C.*, D.*
FROM
  SERVER2.DATABASE1.TABLE1 C, SERVER3.DATABASE1.TABLE2 D
WHERE
  C.COLUMN1 = D.COLUMN1)
UPDATE SERVER2.DATABASE1.TABLE1
SET COLUMN1 = "NEWVALUE"
COMMIT WORK
```

A client-server DBMS can offer four levels of distributed database capability. In order of increasing complexity and functionality, they are remote request, remote transaction, distributed transaction, and distributed request. Only distributed request processing can be considered to support the concept of a distributed database.

Let  
gran  
Be  
own  
As w  
find  
Look  
It's  
W  
ful, a  
gran  
And  
to be  
One  
exce  
New  
• Wit  
gua  
wri  
to s  
ma  
• The  
wit  
the  
• Ma  
to 4  
• We  
and  
Mo  
• We  
be:  
• You  
Plus  
- 30  
• File  
• Un  
• Fil  
• En  
• BR  
• Tol



example, tables from multiple locations to be accessed using a relational join or union operation, as shown in Listing 4.

Of the four types, only distributed request processing can be considered to support the concept of a distributed database. (Of course, other facilities are also required to fully support distributed DBMS, as shown in the box entitled "Distributed DBMS Checklist.") Distributed request processing allows users to distribute data across multiple locations without the application having to know where the data is physically located. The other three types of processing all impose restrictions on what can be done by the application, and often require the application to know the physical location of the data. These other three types do, however, permit access to remote data, and do allow users to perform application processing (client processing) at a different location from the database processing (server processing), i.e., they all support a form of client-server processing. As you shall see, there are several different options for use in developing distributed applications.

### Types of Data Distribution

Data in a distributed environment can be distributed in one of several ways:

- **Manual extract:** where a user causes data to be copied from one location and loaded into one or more tables at another location. Remote request or remote transaction processing could be used to perform a manual extract.
- **Snapshot:** where the DBMS periodically extracts data from one location and loads it into one or more tables at another location. The user defines the frequency (every day at midnight, for example) at which snapshot processing is done. A snapshot is usually created for read-only processing.
- **Replication:** where the DBMS maintains multiple copies of the same table at multiple locations. The location of the replicated data should be transparent to the applications accessing it. The process of keeping each of the copies up to date can be performed either asynchronously or synchronously with the processing of applications that modify the replicated data.

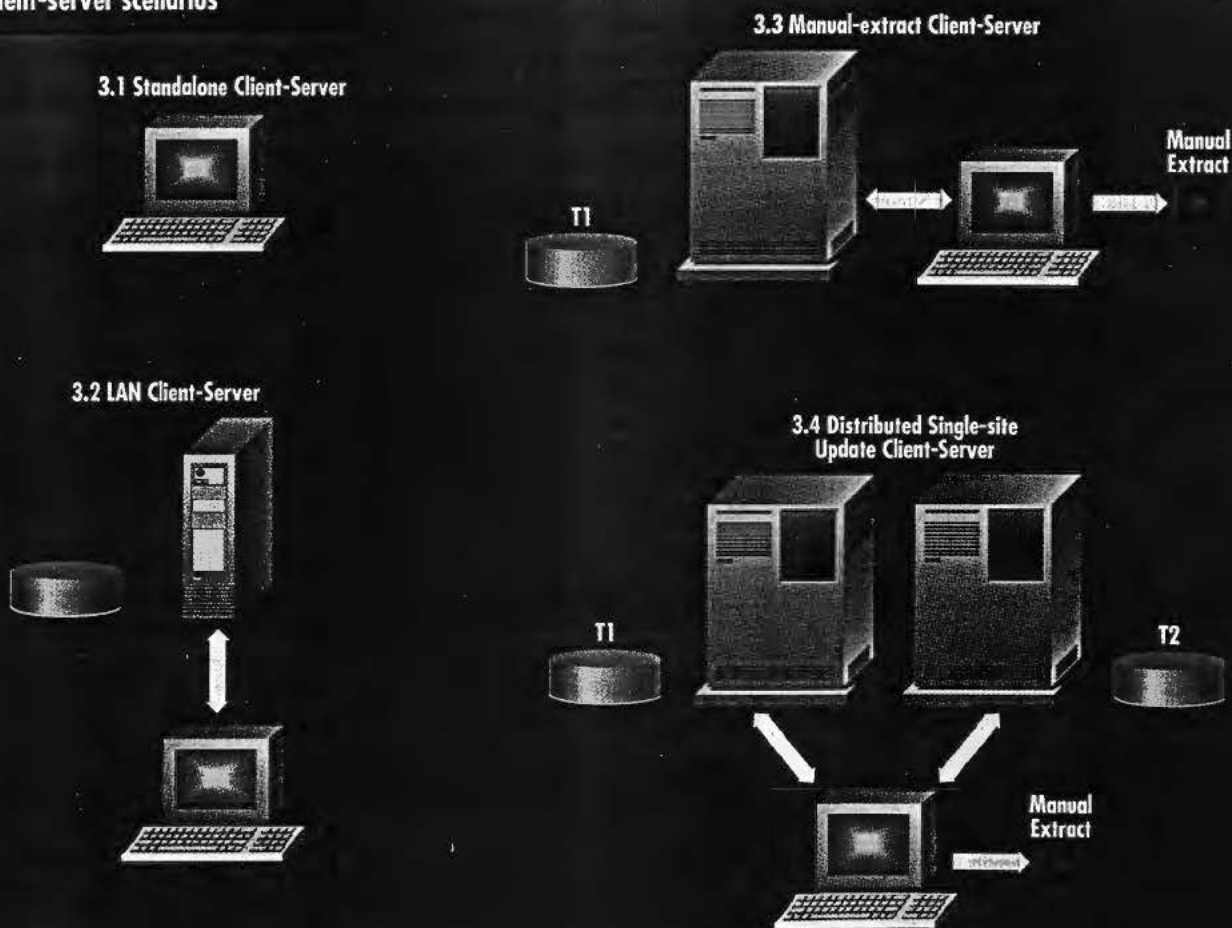
- **Fragmentation:** where a table is broken into multiple pieces, and each piece physically stored at a different location, usually in order to keep data located physically close to where it is used while allowing full access to all the data in the system from remote sites. Applications accessing the fragmented data perceive it as a single table. Fragmentation can be done vertically by subsetting the columns of a table, or horizontally by subsetting the rows of a table.

### Types of Client-server Processing

Having defined the various types of distributed data access and data distribution, we can now look at how these could be used by different kinds of client-server based distributed applications, and review the distributed database architectural features required by each. There are many different types of distributed application. To demonstrate some typical examples, we will use the six scenarios shown in Figure 3.

**Stand-alone client-server.** The kind of client-server application illustrated in Figure 3.1 is one in which the client

**FIGURE 3**  
Client-server scenarios



proc  
on th  
confi  
serv  
cessi  
ever,  
sligh  
the c  
ent a  
funct  
Whe  
on a  
proc  
man  
proc  
cessi  
M  
ure  
appli  
by a  
data  
syste  
by r  
serv  
Th  
can  
cess  
serv

process and the server process reside on the same physical platform. In this configuration, the server can still conserve resources by providing shared processing to multiple applications. However, it usually does so at the cost of slightly degraded performance due to the cost of communications between client and server process. In effect, the functionality is that of a local DBMS. When multiple clients or servers are run on a single hardware platform, multiple processors may be used to improve performance. Nonetheless, neither distributed processing nor distributed database processing are supported in this case.

**Manual extract client-server.** Figure 3.3 shows a style of client-server application in which processing is done by accessing subsets of the corporate data that have been moved to the client system. These subsets will be created by regular manual extracts from the server.

This style of client-server application can be employed by an end user to access corporate data from a single remote server and is normally restricted to read-

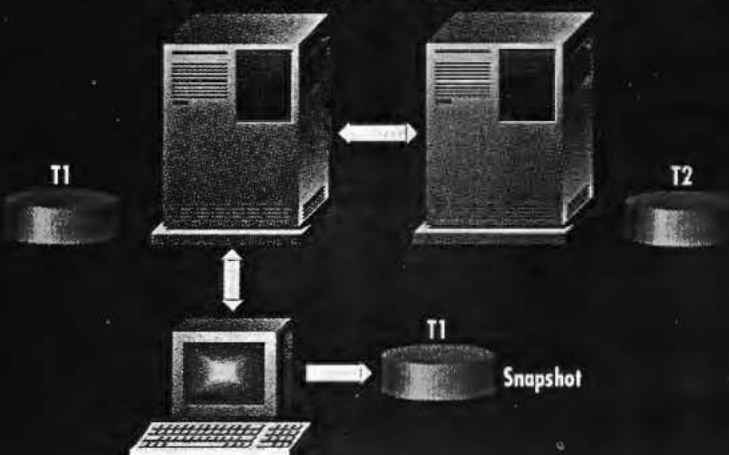
only access. The extract is necessary because corporate data is frequently not in an appropriate format. The end user may, for example, want to see summary information or consolidated data, rather than detailed data; that consolidated data would then be processed locally. The data may be inconsistent while it is being accessed by the end user if corporate applications are maintaining the data on the server at the same time. Dynamic access to data can be done under this architecture using remote request processing or remote transaction processing. The amount of dynamic access typically is kept to a low level, however, because of the potential performance impact on the remote system.

**Stand-alone LAN client-server.** This is the typical client-server configuration on a stand-alone LAN. Multiple client processes — usually one per workstation or PC — are called on to handle the presentation, business, and database logic while the server handles database access. There is a slight disadvantage to this kind of approach as opposed to that of the stand-alone case shown in Figure 3.1: Commu-

nication between client and server is via a LAN link, which is much slower than the shared memory link used in a stand-alone host-based client-server application. This disadvantage may be offset by the additional processing power of one workstation per client process.

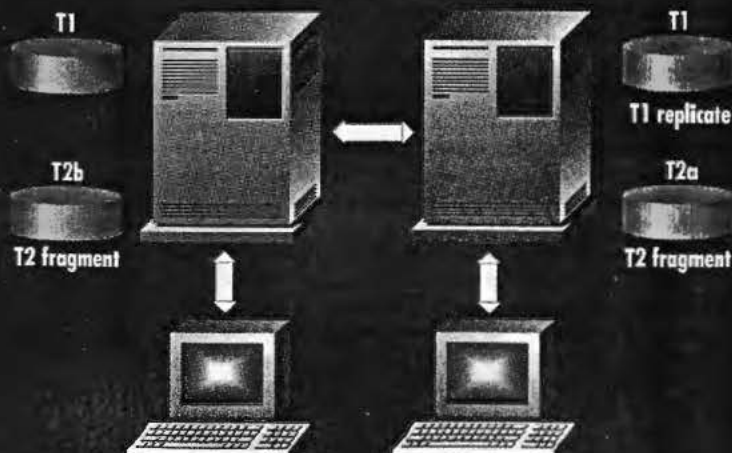
**Single-site update client-server.** This style of client-server application ups the ante to access of multiple remote locations using distributed transaction processing. The data accessed at any one location is usually still independent of the data stored at another. Because the remote server locations are not connected by a network link, neither location can act as a coordinator in a two-phase commit protocol. For this reason, this style of processing only allows a client transaction to update data at one remote location. (Note: if the product provided support for the client to act as the coordinator, multi-location updating is then possible.) Even though a transaction can update data only at a single location, it is still possible for deadlocks to occur, so distributed concurrency controls are required for single-site update client-server

3.5 Distributed Multiple-site Update Client-Server



Of the six different kinds of client-server based distributed applications shown here, only Figure 3.6 supports full distributed database processing.

3.6 Fully Distributed Database Client-Server



applications. Data distribution would normally be handled by manual extracts of data from remote servers.

**Multisite update client-server.** A multisite update client-server application adds support for a two-phase commit protocol between locations and, therefore, allows client transactions to update data stored at multiple remote locations. These extensions permit data stored at one remote location to be related to data

stored at another location, thus providing the first elements of a distributed database capability. With this type of capability in place, data distribution by manual extracts will be replaced by DBMS snapshots.

Of course, this type of distributed client-server application can provide end-user access to remote corporate data just as the previous examples did. This architecture is also suited for engineering and

scientific applications that need to access and maintain distributed data.

**Distributed database client-server.** Finally, we come to a true distributed database application using distributed request processing. Here, the DBMS provides both data fragmentation and replication, allowing faster access for read processing. You should be aware that there are performance implications for significant data modification operations.

## Client-server Applications Development

**T**he three main phases of application development to consider when reviewing the use of workstations for downsizing application development are analysis and design, coding, and testing.

### Analysis and Design

If this phase of the application life cycle is independent of the underlying DBMS, it poses no special development problems from a DBMS perspective. On the other hand, upper CASE tools usually maintain analysis and design metadata such as entity/relationships or object-oriented definitions, data flow diagrams, and structure charts in a CASE data dictionary. Some upper CASE tools are appearing that use either a relational DBMS for storing and manipulating the CASE data dictionary or for which a bridge can be used to move metadata between the DBMS and the CASE tool data dictionary. Dictionary compatibility with the DBMS catalog will become increasingly important as this technology develops.

Workstation based upper CASE tools for doing application analysis and design have existed for several years. In the client-server environment, the server may also be the location of a global repository that may then be remote from the workstation. If a bridge is used to move metadata to or from such a remote repository, issues such as reintegration, versioning, and name resolution become important. To date, no repository exists with a satisfactory mechanism for achieving these tasks.

### Coding

Support for coding of client-server applications varies from product to product. This is particularly true of embedded SQL support. Support for multiple servers from within a single compilation unit is not a standard feature and, if supported, varies in implementation. For example, it may be possible to issue a CONNECT statement (to a particular server) as a SQL extension. Another mechanism is to declare a SQL cursor for a specific server.

During the coding phase of application development, editing can be performed on the workstation. If the host language is supported on the workstation, modules and perhaps the entire application can be compiled as well. SQL statements can be prototyped on a local DBMS provided its SQL dialect is similar to the host DBMS that will be used for production operation. As long as the DBMS supports full location transparency for applications programs, the application can later be redirected to the host server without recompilation.

In addition, stored procedures and triggers are features likely to be supported by DBMSs with a client-server architecture, as these reduce network traffic. If stored procedures are supported by the DBMS server, coding can be substantially simplified. This benefit is not dependent on the whether the

DBMS is local or remote. Similarly, features such as support for referential and other forms of integrity enforcement within the DBMS can reduce network traffic and application coding. All of these features may improve portability by reducing applications code, but unfortunately DBMS vendors typically use different syntax and/or semantics when supporting these nonstandard capabilities.

Finally, application development tools (application generators, source code and configuration management utilities, symbolic debuggers, and so on) that can use a client-server architecture enable application prototyping and development to be done using the DBMS on the server platform. In this case, a separate database is usually set up on the server platform for access during prototyping to avoid both possible loss of data integrity and concurrency conflicts in the production database.

### Testing

Testing a client-server application on a workstation is much easier than testing a host-based application on a workstation. The latter requires that a development tool be capable of simulating the final production environment. While simulating the final production environment can be straightforward if the workstation runs the same operating system (for instance, a VAX/VMS or UNIX System V) as the host, it can become quite complicated if there are significant differences (as there are between MS-DOS and IBM's CICS, for example). The differences might require emulation of transaction monitors, data communications monitors, operating systems services, compiler, file system, and DBMS.

If, however, the corporate application will run on the workstation as a client-server application, the only component that might ultimately have to be tested on the host is the SQL processing. While SQL can be tested against a local DBMS, if the workstation is a single-user environment, some means of emulating network communications, transaction management, and concurrency for test purposes is required. Otherwise these portions of the application functionality must be tested on a host DBMS using client-server processing.

Some client-server DBMS products provide tools for testing against a model database on the local server and then subsequently and transparently connecting to the production database and server without recompiling or relinking the application. This is a powerful test vehicle for applications, limited only by the capabilities of the workstation (single vs. multiuser programs, small databases, memory limitations, and so forth). It allows developers to isolate problems with communications and the network from those involving application functionality, reliability, or correctness. It also provides strong isolation between the development and production environments. ■



Only something like IBM's recently announced high-speed fiber links between remote servers — which allow mainframe channel speeds to be maintained over a distance measured in miles instead of feet — would eliminate concern about this communication bottleneck. Distributed database architecture requirements for this type of application are extended to include location transparency, global optimization, distributed integrity control, and distributed administration.

#### Client-server Communications

There are two ways a client application can connect to a remote server, either directly or indirectly (see Figure 4). A direct connection allows the application to connect directly to the remote server. An indirect connection provides the application with access to the remote server only through connection to a local DBMS. These methods are not mutually exclusive: It is possible that, within an interconnected client-server system, both methods may be used. It is quite likely that the local DBMS accessed by end users will itself implement client-server architecture on a LAN.

With the direct connect approach, the application connects to the remote server through a communications interface. The communications interface is sometimes referred to as a gateway. (Note: This

term has its origins in network terminology and is distinct from database gateways found in the DBMS world.) If the communications interface allows connections to multiple servers, the application could use distributed transaction processing, but this can lead to integrity problems if the application updates data on more than one server. A two-phase commit protocol is required to avoid such integrity problems. Today, remote transaction processing is far more common than distributed transaction processing during access to remote data (and distributed request is rare indeed).

With the indirect approach, a local DBMS handles connections to a remote server through a communications interface on behalf of the client application. With this approach, the client application can access both local data and remote data by making requests to the local DBMS. This might be the approach taken by a configuration in which Oracle Server runs on a local 486-based machine on the LAN that is linked to Oracle running on a larger machine, for example. To handle this kind of approach, distributed transaction or distributed request processing is required.

A simple communications interface is insufficient for either the direct connect method or the indirect connect method if different products are used. In this

case, the client and server or local DBMS and remote DBMS may use different SQL dialects (syntax) and possibly have different semantic behavior as well. Regardless, a piece of software called a database gateway must be developed that attempts to translate all the SQL syntax and semantic differences. Achieving one hundred percent compatibility is an almost impossible task. This imposes the need for sophistication on the part of the user to be aware of and overcome unsolved problems. Unfortunately, the details of this topic are beyond the scope of this article.

#### Client-Server End-User Computing

A client-server architecture provides the opportunity to off-load mainframe processing to a workstation. In a client-server environment, the application processing may be done on a workstation, and a mainframe may be reduced to the role of a database server. Indeed, the mainframe's load may be reduced even further if a LAN-based database server takes on an intermediary role. This will become clearer as we discuss the application of client-server architecture to end-user computing.

One of the factors pushing strongly towards client-server computing and downsizing is the demand for access to data created when a company first installs a

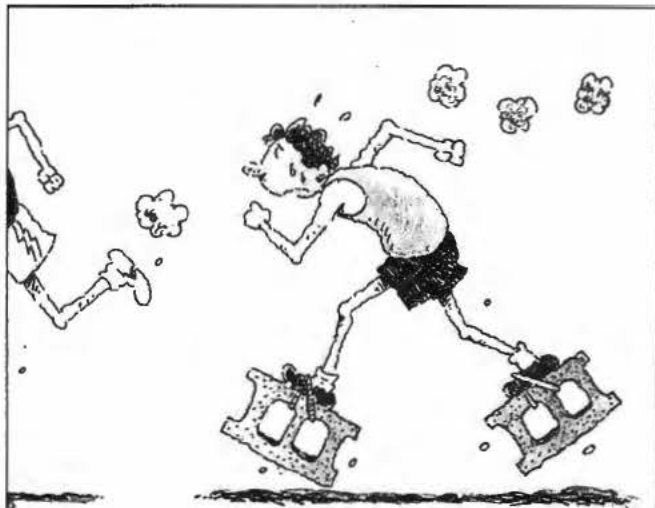


Enhance Your Marketing Plans...

## ORDER YOUR DBMS REPRINTS TODAY!

Reprints are terrific trade show handouts, direct mail inserts and PR pieces. Now you can professionally reproduce your editorial review on 70 lb. glossy paper in four-color, two-color or black and white. Call today for details.

Ellen Hughes  
(415) 366-3600 ext. 401



Is this what it's  
like running complex  
data on your  
relational database?

CIRCLE READER SERVICE NUMBER 98

**FIGURE 4**  
**Client-server connections**

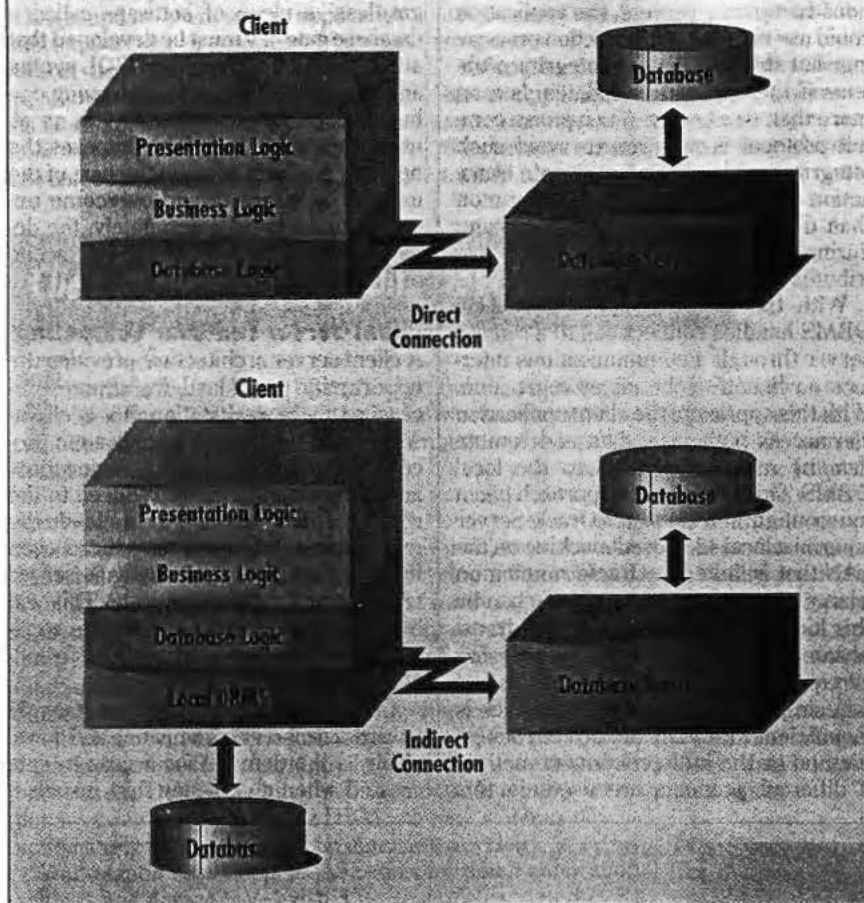


ILLUSTRATION BY MARGARET ANDERSON

There are two ways a client can communicate to the server: direct and indirect connections. With the indirect approach, the client application can access both local data and remote data by making requests to the local DBMS.

new relational DBMS. Many companies underestimate the growth of end-user computing by a factor of two or three when they first install a new relational DBMS. As end users find that they can at last get access to their data, they make more and more use of computing resources. If only mainframe processing power is used, the load created by explosive growth in end-user processing can pose a significant machine capacity problem. Using client-server architecture to offload application processing to work stations saves host computer power and provides a means for cheap, incremental growth. Just as important, it also has the advantage that workstations typically have better tools than their mainframe counterparts. End-users familiar with the GUIs (graphical user interfaces) which these tools offer are unwilling to settle for less sophisticated tools. As one industry specialist has noted, "We are creating a generation of GUI junkies." This situation drives a trend towards the development of workstation-based tools employing windowing technology, further encouraging the move towards client-server architectures.

If the workstations involved are connected to a LAN-based server to which host corporate data can be extracted and downloaded on a regular basis — say, every day at midnight — the processing load on the host is further reduced.

Another problem can be addressed with this intermediary LAN approach. Today most end-user computing on workstations is done in single-user mode. When the end user wants to access corporate data on a host computer, a micro-mainframe link is used to extract the required data from the central database into a file. This file is then downloaded to the workstation over the link. Such copying of data leads to data proliferation and a total lack of control over the extracted data. If the data is instead extracted and downloaded to a LAN-based client-server system every night, a local database server can manage shared access to the data. The data shared by end users will remain consistent and never more than one day old.

This still leaves the situation where an end user on a workstation needs to get up-to-the-moment information from

a host database. In this case, the client-server approach is ideal since it provides dynamic access to the host data. Even so, this access needs to be controlled to prevent performance degradation of the host applications, and also to ensure that the data being accessed is in a consistent state. The direction of the industry for supporting dynamic access to host data by end users is to provide a client-server connection to a host DBMS from workstation end-user tools such as spreadsheets (e.g. Microsoft Excel and Lotus 1-2-3) and query packages.

### Operational Processing

The ability to off-load corporate processing to workstations is a primary goal of client-server architectures. The application specific code and database processing can be run on a workstation using client-server processing against a local DBMS. Alternatively, the application-specific code could be run on a workstation and the database processing could be run either on a LAN-based database server or against a remote host-based database server. The only difference between these client-server configurations is the server platform.

However, the choice of server platform can have an impact on systems management. The choice will depend on capacity and performance requirements, and on the functions (for example, systems management tools, continuous operation, and high availability features) provided by the DBMS server. The advantage of the client-server architecture is that changing the platform on which the server is located should be transparent to applications and need not even be determined initially.

### Configurations

For purely operational reasons, it may be necessary to configure a client-server architecture to support multiple-server connections from within a single application. Even if it were possible and desirable to integrate pre-existing systems transparently into a single distributed DBMS, degrees of location transparency vary from product to product and is less than perfect today. Because of this, there must be a means to configure connection to multiple servers and multiple databases per server. Control of both network and database access must be provided. The degree to which these facilities are provided and transparency supported affect the amount of (typically) complex code that must be written by the developer to compensate for the deficiency.

Environments without fully distributed functionality present a number of problems. For example, multiple-server support does not imply that distributed query processing is supported. A particular SQL cursor is usually restricted to processing



statements that address a single server and any databases that are accessible to it. By contrast, if the vendor assumes that servers are physical entities, a network node address may be used to control a server connection. For this reason, multiple-server support also does not imply that multiple database servers on a single platform can be accessed from within an application.

It may be desirable for some applications to span servers for reasons of security and/or isolation, but for other applications to be confined to accessing a particular server's databases. Sometimes server or just network node access is controlled by a start-up parameter. The value of this parameter might not be dynamically alterable during the run of the application. Such a mechanism may or may not allow access to multiple servers at a given node. On the other hand, there may be no control at all over the server to which a particular request is sent, the decision logic for this being transparent to the application developer and an all-or-nothing decision for the system manager. In this situation any authorized application can access any server or database defined as part of the distributed system and no others.

Multiple client connections to a single-server support require that transaction management and locking within the data-

base engine be handled properly. It also requires that each connection between client and server be identified uniquely and that appropriate mechanisms be provided for applications programs to interact concurrently. Some DBMSs support remote services such as remote procedure calls from one server to another. This allows more than one remote database to be accessed from within the boundaries of a transaction, but it does not mean that the remote service will be committed with that transaction. Thus, distributed transactions may or may not be supported.

### Strengths and Weaknesses

As with most other technologies, there are advantages and disadvantages in using a client-server architecture. The key advantages are:

- The savings in host processing power
- Independent scalability of client and server platforms
- Code modularity via shared services (a server provides access to code that can be shared by multiple applications)
- The ability to use workstation end-user and development tools.

With the move towards increasingly more sophisticated and user-friendly workstation tools, the last of the four will become

the dominant reason for using a client-server architecture.

Improperly used, the client-server architecture can cause some difficulties. These potential weaknesses are:

- The impact of distribution on performance
- More-complex information systems management

We discuss these potential drawbacks in more detail below.

### Performance Considerations

The key to good client-server performance is to improve the efficiency of the transmission of database requests and result data across the network. Client applications communicate with a remote database server using a database language such as SQL. A database server, after processing an SQL client request, sends back to the client only the data that satisfies the request. This is much more efficient than a file server architecture, where the complete file is sent from the server to the client.

The set-level processing aspect of SQL also aids performance. A client application can, with a single SQL statement, retrieve or modify a set of database server records, rather than having to issue separate sequential requests for each desired

## INSTALL, BAT

First impressions last.

### INSTALL 3.0 gives your software product a professional introduction.

Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.

INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.

- Features very sophisticated error-handling
- Does high performance data compression
- Can safely and reliably modify system files
- Allows selective installation of parts of your program
- Complete source code included
- No royalty charges
- Free technical support

INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.

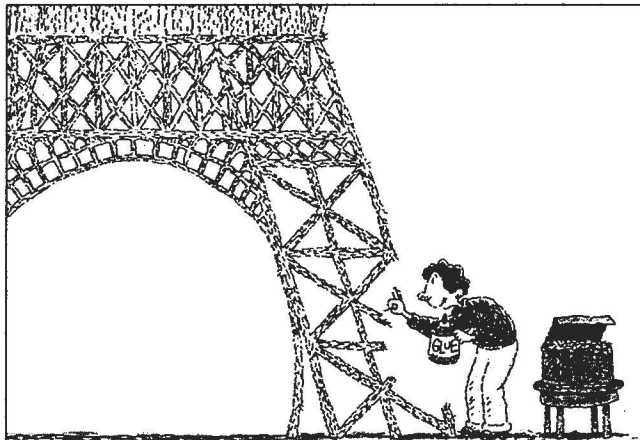
INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.

**KNOWLEDGE DYNAMICS CORP.**

Highway Contract 4, Box 185-H  
Canyon Lake, TX 78133-3508  
MasterCard/VISA/COD/POs welcome

**\$249.95 + shipping**

Money back guarantee  
Sales 800-331-2783  
Intemat'l 512-964-3994  
24 hr FAX 512-964-3958  
24 hr BBS 512-964-3929



Does this remind you of what it's like building an application to manage complex data?

CIRCLE READER SERVICE NUMBER 250

CIRCLE READER SERVICE NUMBER 107

record of each of the base tables, as in older database systems. Client-server SQL statements work most efficiently when doing data modification, because data results need not be sent over the network. However, because SQL can create a results table that combines, filters, and transforms data from base tables, considerable savings in network data communication are effected even for data retrieval. Only the rows and columns of data needed by the application need to be sent over the network.

On the other hand, network costs cannot be ignored when doing distributed or remote processing of any kind. Even though a relational DBMS reduces the amount of requests and data that must be sent over the network, the speed of the network plays a major role in the transaction response times seen by the workstation user. Network protocol software must handle the information at either end of the communications link and use of a network operating system is not unusual. Network operating systems use

both host and client computer processing power.

In the case of a host-based client-server implementation, the processing power required by the network operating system (for example, VTAM in an IBM mainframe environment) may or may not be offset by the processing power saved by off-loading the application processing to a client workstation. If network processing costs on the workstation are too great, performance is degraded for the application processing. On the other hand,

## Distributed DBMS Checklist

**A** true distributed database system requires the following architectural features:

**Location transparency:** The physical location of a table when doing application processing is handled by the distributed DBMS. Neither users nor applications need have information about the physical location of a table. This feature is particularly important when doing distributed request processing, and when accessing fragmented or replicated tables.

**Global optimization:** The relational DBMS optimizer takes into account the cost of accessing remote data when determining data access paths. This feature is important for good performance when doing distributed request processing.

**Distributed commit:** The system uses a two-phase commit protocol when updating data at multiple locations. This

feature is required for distributed transaction and distributed request data modification operations.

**Distributed concurrency control:** The system uses a global concurrency mechanism to control multiuser access to data at multiple locations. This facility is required when doing distributed transaction or distributed request data modification processing.

**Distributed integrity control:** The system ensures that the distributed database integrity rules (referential constraints, for example) are enforced.

**Distributed administration:** Facilities are provided to define, create and maintain tables in a distributed environment. The DBMS should also have tools to monitor and tune the distributed database system. ■

# Now there's ONTOS. All the speed of files with the productivity of a relational database.

Who says that you can't have your cake and eat it, too? Certainly not Ontologic. Not when they have ONTOS. ONTOS from Ontologic is an object database. And it combines the functionality of relational databases with the speed of files to give you unprecedented productivity.

So now applications developers can have the superior performance of files with the benefits of relational databases like built-in multi-user sharing, integrity, SQL, transactions and concurrency control.

ONTOS. Proven at sites worldwide. For more information call 617-272-7110 or return the coupon.

Please have a salesperson call.  Send me more information on ONTOS.

Mail coupon to Ontologic, Inc., Three Burlington Woods, Burlington, MA 01803.

Name \_\_\_\_\_ Title \_\_\_\_\_

Company \_\_\_\_\_ Phone \_\_\_\_\_


Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_

Zip \_\_\_\_\_

Current O/S \_\_\_\_\_

Current Hardware \_\_\_\_\_



The Experienced Object Database Company.

DBMS 11/90

CIRCLE READER SERVICE NUMBER 122

presentation logic for advanced graphical user interfaces (GUIs) can easily consume mainframe resources if workstations are not used.

Clearly, network processing costs are significant in determining the performance of client-server applications. In evaluating the potential network load, the number of transactions per second and the volume of returned data must be taken into account.

With some DBMSs, the number of SQL statements being sent across the network by client applications can be reduced by storing a named group of related SQL statements and associated program logic in the database server as a stored procedure or stored program. The stored procedure can then be executed by a single request from the client. This approach also has the advantage that the stored procedures can be shared by multiple client applications. The problem with this technique is that there is no agreed standard for defining or invoking stored procedures (the ANSI SQL committee is considering such a standard). Furthermore, not all DBMSs support stored procedures. For example, DB2 and Oracle do not support them at the present time while Sybase and Ingres do.

Nonetheless, SQL and stored procedures help reduce network traffic compared to that with non-relational access (which retrieves entire records from each file or table), the rows of a results table still have to be sent over the network from the client to the server. Most relational applications use a cursor to fetch data, one row at a time, across the network. This involves considerable communications overhead. It can be reduced if the results are sent in blocks across the network. The result data can then be stored in a buffer on the client workstation and processed using a SQL cursor without further network interaction. A problem arises, however, if the client application wants to update the retrieved data. If the database server has not locked the set of result rows, an integrity exposure exists because other applications could have updated the data since it was retrieved. Possible techniques for solving this problem are:

- to use blocking and lock the complete query result on the server until the client issues a commit,
- to use blocking and lock each row on both the client and the server as it is fetched by the client application,
- not to use blocking, and instead to send the data across the network to the client, one row at a time, locking each row as it is fetched by the client application, or
- to use an optimistic concurrency control mechanism that checks for update collisions at commit time.

## Information Systems Management

In a centralized development and operational environment, all our programs, data, and data definitions are stored in one place. When we distribute application development, application processing, or data, the simplicity afforded by such centralized storage and control is lost. For example, new problems include:

- Management of multiple program libraries
- Management of multiple data definitions
- Database monitoring and performance tuning
- Backup and recovery of a distributed database
- Network management

These problems have an impact on the personnel and operations as well. Suffice it to say that this area must be considered when implementing client-server applications.

## Conclusions

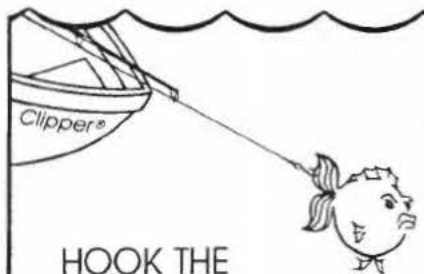
In this article we have briefly outlined the features of a distributed database system and have discussed several different types of distributed applications. We have also shown that you do not necessarily require full distributed database support to build distributed applications.

Client-server implementations that fall short of full distributed database support are nonetheless very viable. For example, client-server architectures involving manual extraction or snapshots, or that limit updates to a single SQL statement or to a single site, clearly impose restrictions on what the application can do. If location transparency or server scalability isn't provided, the application has less ability to withstand change. Nonetheless, these variations of client-server architecture can provide substantial benefits to organizations until some of the more advanced technology that a fully distributed database demands is in place.

## Acknowledgments and References

Much of the material contained in this article was taken from articles published in *InfoDB* magazine (references 2 and 3), which discuss specific issues in more detail. They are part of an on-going series of detailed discussions on distributed architectures, information architectures, gateways, and so forth.

1. C.J. Date. What is a Distributed Database System? *InfoDB*, Volume 2, Numbers 2 and 3, 1987.
2. C.J. White. Using Distributed Database: Application Types, *InfoDB*, Volume 4, Number 1, 1989.
3. C.J. White. Client-server Computing in a DB2 Environment, *InfoDB*, Volume 5, Number 1, 1990.



HOOK THE  
PERFECT INTERFACE!

**GrumpMenu** is the last word in menus for Clipper. If you can create a text outline file, you can create a gorgeous, easy-to-use front end menu with GrumpMenu — it's that easy. Store your menu structure in a text outline file, and Grumpfish Menu instantly generates optimized, ready-to-compile Clipper source code (compatible with Summer '87 and Clipper 5.0)!

GrumpMenu supports six different menu styles, including pull-down, cascading 1-2-3, and boxed 1-2-3. You or your users can change menu style and color "on the fly" from within your application without recompiling or relinking! You can set up configuration files so that each user has their own interface. GrumpMenu's prototype switch will save you hours in developing quick prototypes to show clients and users.

*\*Not only has GrumpMenu given our applications an elegant colorful interface, it has cut our programming time by 20%. We love it, and our users love it!*

— Nicholas Ivon

**GrumpMenu sells for just \$199 US (add \$7.50 US, \$20 international s/h). You can't go wrong with the Grumpfish 30-day money-back guarantee, 30 days free phone support and one year free BBS support (a \$50 value). Improve your profile — pick up the phone and hook yourself the perfect interface today!**

**Grumpfish, Inc.**  
Box 17761, Dept. 30  
Salem, OR 97305 USA

Tel. (503) 588-1815  
Fax (503) 588-1980  
BBS (503) 588-7572