THE MAGAZINE FOR SYSTEMS AND SOLUTIONS DEVELOPERS

UNIX REVIEW

AUGUST 1989 $3.95

DATABASE GATEWAYS: FORM AND FUNCTION

SHARED LIBRARIES IN UNIX SYSTEMS

UNIX FOR THE 386: ISC, SCO COMPARED

THE BIG PICTURE

In order to make effective business decisions, users today need access to information located on remote data management systems supplied by multiple vendors. Significant technical issues must be resolved before users can have true plug-and-play access to data.

BY M. BISHOP and E. WASIOLEK

If UNIX is ever to be widely used in the commercial arena, it must first become a bona fide participant in corporate America's data processing environments, in which IBM mainframes and DEC minicomputers currently hold sway. As straightforward as this assertion is, it ushers in a host of complex implications. Among them is one on which we'll focus our attention here: in corporate data processing environments, UNIX applications must offer so-called "seamless" access to information stored in foreign databases—such as IMS, DB2, or RDB—located on remote mainframes.

In order to make effective business decisions, users today need access to information located throughout their organizations. They should be able to gain this access directly from their desktop machines, whether they are connected via a workstation, a PC, or a timesharing terminal. Yet the bulk of corporate information is controlled by data management systems supplied by multiple vendors. Often, different hardware platforms, operating systems, or database management systems (DBMSs) are implemented within a single company. As a result, one department is cut off from information held by another. This leads to low productivity, organizational inefficiency, and ill-informed management.

Moreover, the fluidity of today's business environments and rapid technological progress require that information management strategies be modifiable. Users do not want to be locked into using a single vendor's machine, network, or database system. Ideally, applications and tools from many vendors should work transparently with servers supplied by several database companies.

Since they know what the problem is, why don't the database companies get together and come up with a solution? As you might expect, the answer is not that easy. Significant technical issues must be addressed before users can have true plug-and-play access to data managed by different database systems. These issues include incompatible client-server protocols, inconsistent query languages, different data types, proprietary error codes, and divergent system catalog structures. Let's consider each of these in more detail.

Proprietary Client-Server Protocols. Most relational database products follow a client-server model of operation: applications (clients) access data by communicating with an SQL engine (server). Very often, this communication is hidden from developers in a low-level protocol designed to facilitate the transfer of queries to the server, and to return data to the client. Problems arise because no two protocols are alike. Vendor A's client cannot establish communication with Vendor B's server, because Vendor A's client sends information in a different order, using a message format different from the one Vendor B's server expects. Once a client application is built, a proprietary client-server protocol is embedded in the binary. Thus, the client application becomes tied to a particular vendor's database engine.

SQL Dialects. Nearly all RDBMSs support Structured Query Language (SQL), but each vendor's SQL version is slightly different. Although an American National Standards Institute (ANSI) committee was established a few years ago to come up with an industry-standard SQL, the database industry is increasing its demands on SQL much faster than the ANSI committee is able to set standards. Vendors routinely implement extensions to the existing ANSI SQL standard in response to requests from customers. In addition, the semantics of the SQL standard are not complete. While SQL syntax is well understood, any given SQL statement may still be subject to semantic interpretation.

The semantics of the ANSI SQL standard are not yet complete. While the syntax of the language is well understood, any given SQL statement may still be subject to semantic interpretation.

The existence of incompatible SQL versions clearly has an affect on application portability. The bulk of today's sophisticated production database applications use embedded SQL for data access. Applications written for a specific SQL dialect and used to access data in a specific database management system must continually be modified if they are to be able to access data across the range of SQL engines available today.

Incompatible Query Languages. The need for a single query language for accessing heterogeneous databases goes far beyond the problem of SQL dialects, since users need access not only to RDBMSs, but also to non-relational DBMSs, which don't use SQL. Many users also need access, for that matter, to non-DBMS data sources such as file systems, where a great deal of corporate information is stored. In the absence of a common query language, applications must use radically different access methods to read the information they need.

Database Error Messages. Not all SQL implementations are the same, then, and non-relational query languages differ even more. Is the same true of the error messages returned by servers to client applications? On this question, the news for developers is not good: each RDBMS supports literally thousands of error messages, and they are not standardized.

This means that client applications written for one vendor's database server cannot access data managed by another's unless they are modified to handle a whole new set of error messages. To access information controlled by five or six different servers, an application would have to handle five or six incompatible sets of these messages. Adding such capabilities to an application is not a good use of programmer time.

System Catalogs. The system catalog is that portion of a DBMS that describes the structure of the database—what tables, fields, and columns it contains, and what the data looks like. Relational database applications access the system catalog primarily to avoid the necessity of hard-coding the "look and feel" of the data inside applications themselves, so that the structure of the data accessed may change without affecting the applications.

As you may have guessed by now, each DBMS vendor's system catalog is unique. The names, structures, and fields used in catalog tables are not consistent across RDBMSs, and non-relational DBMS catalog structures are radically different. Many vendors choose to use an object-oriented representation of forms/frames within the data dictionary, generally in a proprietary format. This makes it quite difficult to port applications from one DBMS to another.

Not surprisingly, there have recently been a number of attempts to address the larger issue of *open data access.* A few of the more noteworthy are described below.

Multiple Software Versions: The Brute-Force Method. It is certainly possible to build multiple versions of application code, each tailored to interact with a particular vendor's data management system. This approach has been taken by Unify Corp., Information Builders Inc., and Progress Software Corp. It provides customers with wide availability on a large number of platforms, if the vendor can afford the porting, testing, and support costs that go with it. While some vendors may have solved the access problem for their customers this way, in so doing they've created a nightmare for themselves. The same executable cannot access data located in different sources. These vendors are therefore spending a good portion of their time porting, modifying code, and testing in order to support a wide range of incompatible DBMS products. It's a risky venture at best.

Vendor-Specific Client Access to Remote Data. Some companies have decided to design and build their own protocols as a way of offering access to multiple data sources from a specific client platform. CL/1, developed by Network Innovations (a wholly owned subsidiary of Apple Computer), is an example of this approach. CL/1 offers Macintosh users access to a predefined set of SQL databases, but does not, however, solve the critical problems associated with differences in system catalogs. Because CL/1 is engineered for a specific set of data management systems, it has to be modified to allow access to new ones. Furthermore, CL/1 client applications are not widely available outside of the Macintosh environment.

"Pass-Through" Data Gateways. With this approach, client applications can in fact get to other DBMS systems, but only by attaching themselves to a proprietary server. Good examples of this architecture are Sybase's Open Server and Oracle's SQL*Connect products.

Under the Sybase scheme, a client application attaches itself to a Sybase server that contains a non-Sybase database procedure. The procedure holds all the code required to access the foreign database: the error-handling, data-reformatting, and catalog-access routines. The procedure is essentially, then, a data gateway constructed for a predefined query. This approach allows client applications to remain portable, because

foreign database code is stored in the server's procedure. It requires, however, that a proprietary server always be present, that code existing on the foreign database platform understand the procedure call, and that a very complicated *gateway procedure* be written for each foreign database. This method allows data to be qualified on the non-Sybase host and downloaded to the proprietary Sybase server. Users are locked into using an SQL engine that may not be their first choice for a number of reasons—performance and feature set among them. In addition, predefined procedures mean that no ad hoc querying of the foreign database is supported.

Oracle's gateway product, on the other hand, allows Oracle client applications to access a DB2 or SQL/DS database (see "Rites of Passage", p.48). The Oracle approach involves hard-coding connections between clients and foreign data managers, making the clients unportable as well as difficult to code. It allows a single client application to use different query languages to access different databases, rather than providing a single query language to access multiple DBMSs. It offers no ability to access data in non-relational databases or file systems, and offers no common access to foreign database catalogs.

In summary, all of these approaches solve some of the technical issues involved in transparent data access, but none of them solves all or even most of the issues. Each requires a great deal of work on the part of developers to gain access to each additional data source.

If user applications are going to be able to access all the data residing in different formats in today's corporations, they will need to have a single interface that is machine-, network-, and data-independent. To meet with wide acceptance, such an interface would need to be based on industry standards. The goal of database vendors is to offer developers the ability to develop a single version of an application that can access any type of data anywhere, without any recoding (this is the meaning of transparency). Another goal is to allow users to purchase a single executable that can access any type of data in any machine location on a network. In both cases, the major objective is to allow applications to maintain data access even when data changes locations or is moved to a different type of database.

In its new release of the Ingres database product (Release 6), Relational Technology Inc. (RTI) includes three components that yield data-type independence for applications, and that specify a consistent data interface, a common query language, a consistent communications mechanism, and common-to-proprietary-construct-mapping processes (gateways) that reside on foreign database nodes. Ingres Release 6 also provides complementary location transparency, allowing the data-access solution to extend across multiple machines and network types.

RTI's Open Data Access is aimed at providing two capabilities critical to information management in today's computer environments: *data accessibility* and *application portability* across heterogeneous databases and file systems (see Figure 1).

Universal accessibility, if attainable, would mean that users and developers, by adding the appropriate database gateways, would be able to use or develop applications that could, without modification, access data in incompatible relational and non-relational databases and file systems. Such applications would be developed once and only once, and they could access data residing in any database for which a data gateway had been provided. *Universal portability* describes the hypothetical situation in which developers have the ability to port a given application to different platforms and run them on top of different types of databases with minimum code rewriting, thanks to the implementation of a consistent data interface.

The key to open data access is data-type transparency. It is made possible through the specification of a layer of data transparency—a common client application interface to all data. The interface implemented by RTI includes a common query language and set of data types, a generic set of error codes, a common system catalog specification, and a consistent communications protocol. All the proprietary features of underlying data managers are hidden by data gateways, which translate proprietary features into their common application interface counterparts—DB2-specific errors into generic errors, for example, or IMS catalog structures into common system catalog structures, or Open SQL to RMS-indexed file-access routines. Seen from the client-application side, the data source being accessed always appears to be a relational database of the same type. The underlying foreign data manager always "thinks" that it is being accessed by a client of its own type. Without the specification of a consistent client interface to all data, data-type-independent application development would not be possible.

Open SQL is a query language that applications may use to access relational, non-relational, or file-system data (see Figure 2). RTI developers have composed the document defining this greatest common denominator of the SQL variants used by the industry's major RDBMSs: DB2, RDB, Tandem, and the ANSI Level I Standard (SQL-1). All Ingres Release 6 products generate Open SQL, and all Release 6 tools allow for the building of Open SQL, vendor-independent database applications. Applications developers who use only this set of SQL statements should be able to design applications that are portable to all relational, non-relational, and file system data gateways that implement Open SQL. Developers can choose to use "portable" SQL statements, or to use the dialect of a particular DBMS product. While Open SQL allows a "pass-through" mode to execute proprietary data manager statements directly, application portability is lost if this mode is used.

DBMS products that use Open SQL and define a reasonably large set of data types commonly used by the industry (integer, character, floating point, and others) are a first step toward implementing a much broader set of functionality in the language itself.

A key component of the Open SQL language specification is the handling of error messages. Using the ANSI Level II SQLSTATE error specification as a model, developers at RTI have modified Ingres so that it supports a minimal set of "generic" error messages. Developers can now code applications using this set of about 40 messages instead of the more than 3,000 DBMS-specific error messages that are usually supported in an RDBMS product. For DBMS-specific error handling, each generic error message has a proprietary error code attached.

Developers can also count on the appropriate Ingres gateways to map proprietary database error codes, providing maximum applications portability across and accessibility to other databases.

To resolve the question of incompatible system catalogs, RTI has defined a uniform set of database views for delivery with its RDBMS products. Once applications developers use them, all system catalogs look the same to the program, whether it is running with Ingres or another database system. These views present a consistent look at tables, forms, columns, and entities, freeing the developer from having to know the

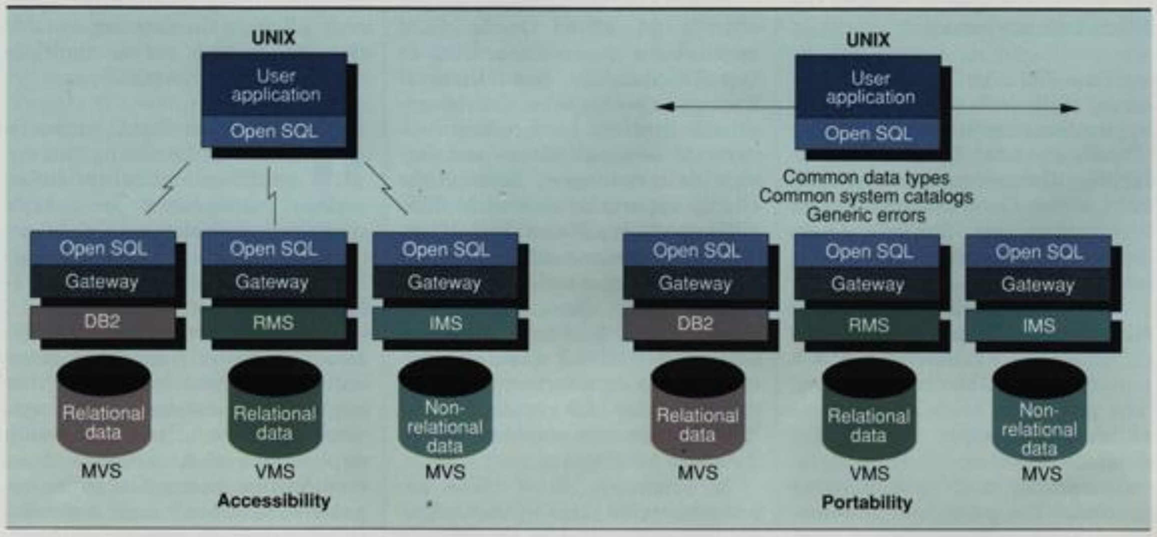


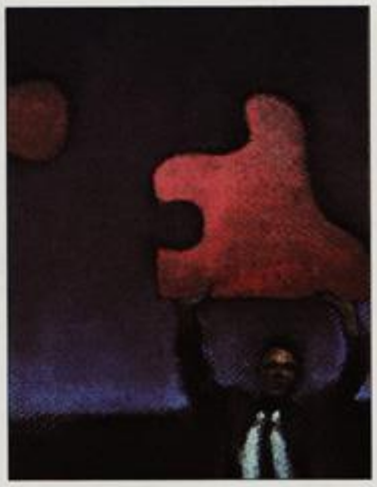
Figure 1 — Two capabilities are critical to information management in today's computer environments, with their heterogeneous databases and file systems: *data accessibility* and *application portability*. Using the Open SQL, GCA, and gateway components shown above with an additional networking component, applications located on a particular machine can access data stored in different databases at different locations. In addition, applications can be ported, with minimal recoding, to other platforms running different databases.

specifics of a particular DBMS.

For obvious reasons, corporate MIS directors are keen on protecting their investments in data stored under proprietary data management systems—be they relational DBMSs such as DB2, SQL/DS systems, or RDB; non-relational DBMSs such as IMS/DB; or non-database data sources such as DEC's RMS file management system or IBM's VSAM.

One way to provide access to proprietary data sources is to provide database gateways, which are programs that run on a host machine that holds a foreign database. The design rule RTI used in putting such gateways together was that application code must operate identically whether it is accessing data in a native database or data stored in a foreign format.

Ingres data gateways are independent processes that map consistent constructs into the proprietary equivalents on the underlying data manager. Open SQL statements sent by a client application to a proprietary server are intercepted by the data gateway process and translated into the appropriate proprietary access language. Open SQL statements might be translated into, for example, an RMS indexed file access routine. The proprietary server then executes the access routine and returns data to the gateway process. The gateway translates proprietary data types to Open SQL data types—a DB2 gateway, for instance, translates a DB2 data type into an Open SQL data type. If an error occurs, the gateway translates the proprietary error code sent by the server into a generic error and prepends it to the proprietary error code before returning the error code to the client application.



If the client application needs access to the proprietary server's system catalogs, the client application accesses the common system catalog, which is then translated into a proprietary catalog-access routine. The data gateway also handles all protocol conversions and message format translations between incoming message formats and protocols and outgoing, proprietary client-server protocols and message formats.

Differences in the query languages, data types, error messages, and system catalog structures used by underlying data managers are all, therefore, hidden from client applications. No modification of client-application or DBMS code is necessary to ensure access to data in radically different data sources.

All Ingres Release 6 products use a common protocol to communicate with each other. This protocol, the Global Communications Architecture (GCA), is based on the International Standards Organization (ISO) Remote Data Access (RDA) specification, which defines how client applications can access remote databases. Ingres application developers use GCA without necessarily knowing about it, since it is automatically embedded into all Ingres applications at compile time. Once an application is built, it can "talk" to all other Ingres products and to any other DBMS that understands GCA. As RDA gains more commercial adherents, the developers and users of database systems will indeed have a wider range of SQL engines and tools to choose from. GCA works transparently on top of industry-standard OSI communications protocols such as TCP/IP, SNA, and DECnet.

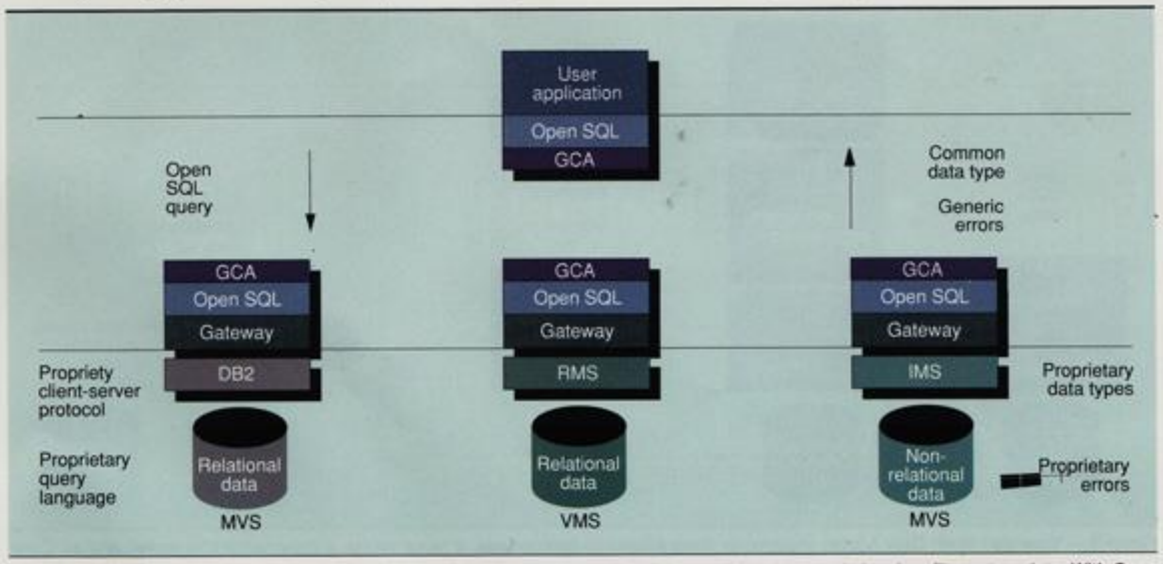


Figure 2 — Open SQL is a single query language that applications may use to access relational, non-relational, or file-system data. With Open Data Access, user applications have a consistent interface to varied data managers. Applications use Open SQL and are returned data in common data types. If an error occurs, a single error set is handled. Client-server communications are enabled by a common protocol, GCA.

In reality, transparent access to heterogeneous data sources is not all that organizations require today—location transparency is required as well. Not only should client applications be able to access information in multiple types of data sources, but they should also be able to access information in different locations without the applications themselves having to be modified. Data should be able to move between data sources—that is, be downloaded from one type of database to another—without requiring that client applications be modified.

In order to accomplish these goals, RTI's Open Data Access solution implements three additional components: a GCA name server, a GCA communications server, and an Open SQL/GCA-based distributed data manager (see Figure 3).

The GCA name server provides database location transparency for user applications: a user application specifies the database name it wants to open, and the name server maps that name to the process ID of the Ingres data manager, data gateway, or distributed data manager responsible for managing the database named. If the target database is remote, the name server works with the GCA communications server to connect to the remote database server.

The GCA communications server provides transparent access to remote data. It accepts Open SQL requests in GCA message format and forwards messages to the appropriate remote system communications server, which then, through the local name server, attaches and sends the message to the destination Ingres or foreign data manager (through an Ingres gateway). The communications server is an independent, multithreaded process that routes Open SQL requests from multiple local clients to multiple remote servers using multiple communications protocols, and concurrently processes requests coming in from multiple remote clients to multiple local data managers. It is based on the upper four layers of the OSI reference model and is both network- and machine-independent.

Are truly portable DBMS applications possible? This question is being asked with increasing frequency by information management professionals. Just a few years ago, the answer to it would have been distinctly noncommittal. Today, the situation has markedly improved.

Challenges lie ahead, however—not the least of which is the need for DBMS companies to cooperate on standardizing key pieces of technology. The fact that Open Data Access is based on the SQL, RDA, and OSI standards might help speed general industry acceptance. To the extent that other vendors' databases utilize a common method, gateways will no longer be needed. However, non-relational DBMSs will probably never be rewritten to comply with new standards, so non-relational gateways will probably never disappear.

The good news is that companies are cooperating. Standards bodies such as the Open Software Foundation and ISO are studying database interoperability issues, customers are driving DBMS vendors toward solutions, and real products that address many of the technical issues are beginning to appear. Although it may take some time, the future for standard, portable, transparent DBMS applications looks very bright. ●

Mitch Bishop is director of marketing for UNIX-based INGRES products at Relational Technology Inc. Previously, he was director of operating systems development at Altos Computers.
Eric Wasiolek is RTI's manager of distributed INGRES product marketing, with responsibility for GCA and networking products as well as for the INGRES/STAR heterogeneous distributed data manager.

Acknowledgement

Dave Simonson of RTI is the orginal architect of INGRES Open Data Access.

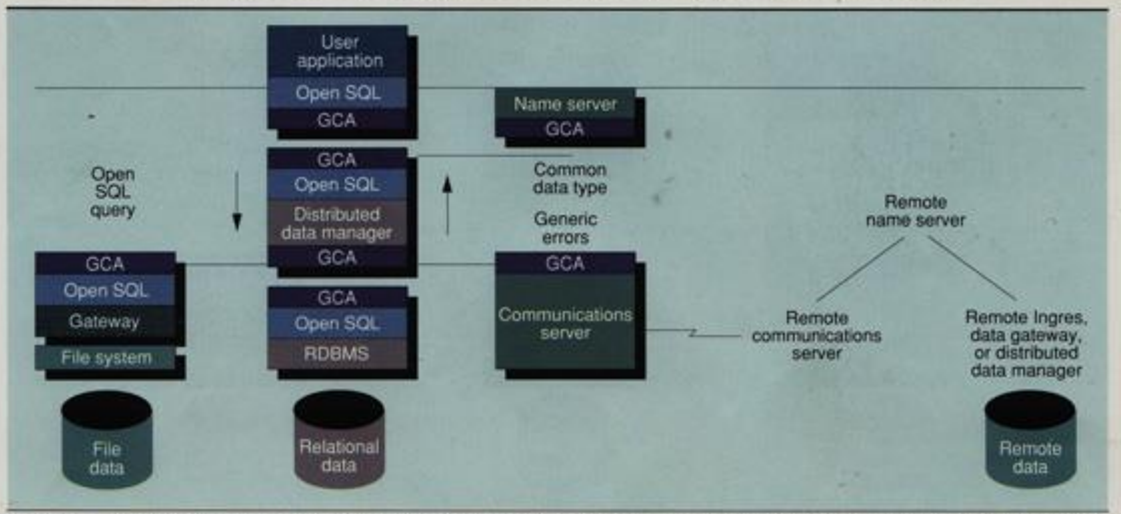


Figure 3 — Extended Open Data Access implements three additional components: a name server, a communications server, and an Open SQL/GCA-based distributed data manager. The GCA name server implements a local database location service. Used in combination with the communications server, it can also provide transparent access to remote heterogeneous data. The Open SQL, GCA-based distributed data manager allows a single query access to information in many types of databases and file systems in multiple locations on a network.

Reprinted with permission from UNIX REVIEW
© 1990 MILLER FREEMAN PUBLICATIONS.