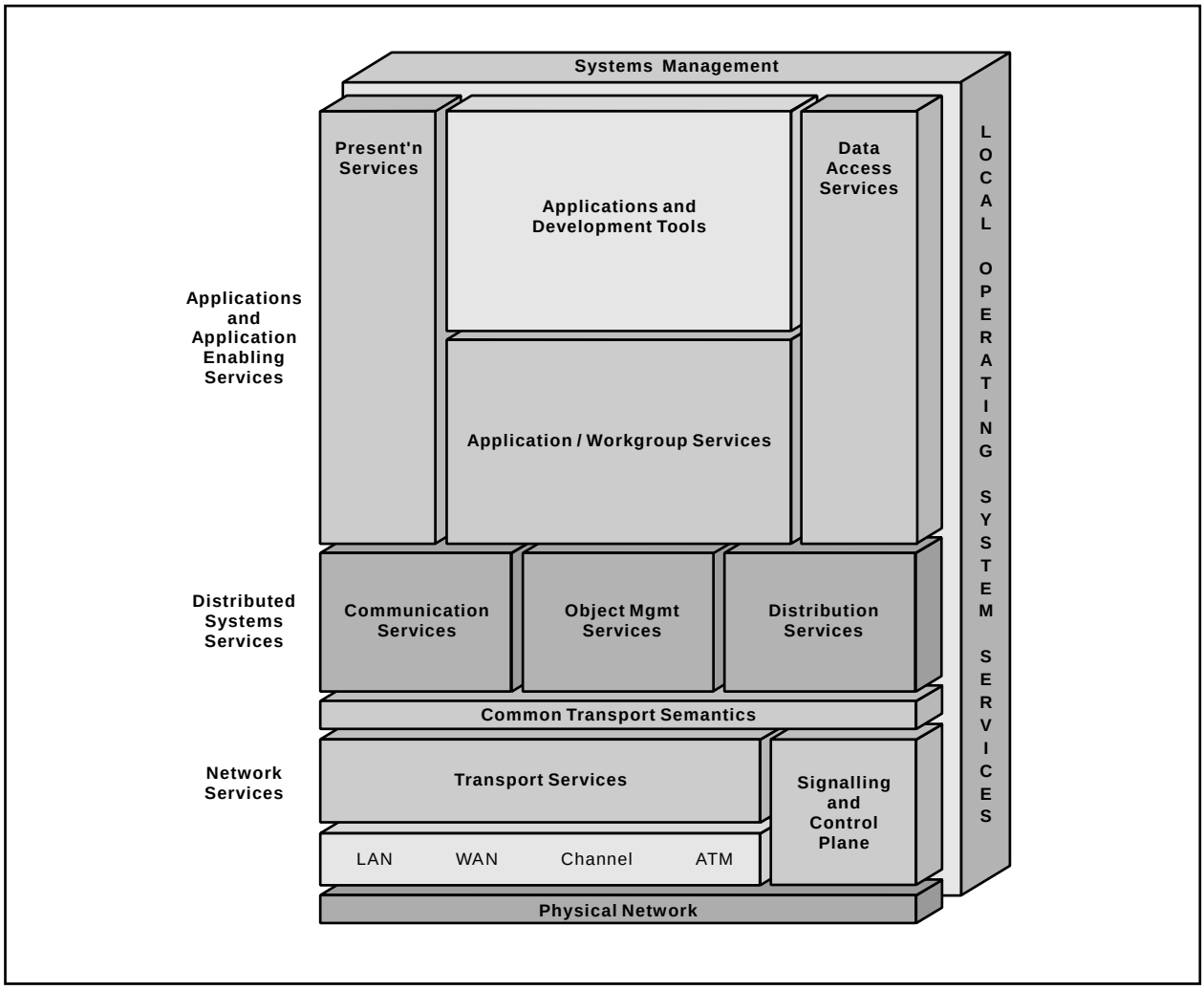
Open Blueprint

Relational Database Resource Manager



GC23-3925-01

Open Blueprint

Relational Database Resource Manager

GC23-3925-01

About This Paper

Open, distributed computing of all forms, including client/server and network computing, is the model that is driving the rapid evolution of information technology today. The Open Blueprint structure is IBM's industry-leading architectural framework for distributed computing in a multivendor, heterogeneous environment. This paper describes the Relational Database resource manager component of the Open Blueprint and its relationships with other Open Blueprint components.

The Open Blueprint structure continues to accommodate advances in technology and incorporate emerging standards and protocols as information technology needs and capabilities evolve. For example, the structure now incorporates digital library, object-oriented and mobile technologies, and support for internet-enabled applications. Thus, this document is a snapshot at a particular point in time. The Open Blueprint structure will continue to evolve as new technologies emerge.

This paper is one in a series of papers available in the *Open Blueprint Technical Reference Library* collection, SBOF-8702 (hardcopy) or SK2T-2478 (CD-ROM). The intent of this technical library is to provide detailed information about each Open Blueprint component. The authors of these papers are the developers and designers directly responsible for the components, so you might observe differences in style, scope, and format between this paper and others.

Readers who are less familiar with a particular component can refer to the referenced materials to gain basic background knowledge not included in the papers. For a general technical overview of the Open Blueprint, see the *Open Blueprint Technical Overview*, GC23-3808.

Who Should Read This Paper

This paper is intended for audiences requiring technical detail about the Relational Database Resource Manager in the Open Blueprint. These include:

- Customers who are planning technology or architecture investments
- Software vendors who are developing products to interoperate with other products that support the Open Blueprint
- Consultants and service providers who offer integration services to customers

Contents

Figures

Summary of Changes

This revision describes these significant updated items:

- How Relational Database relates to network computing
- Relational Database support for Mobile Computing
- Relationship of Relational Database to Collaboration Resource Manager
- The extension of DRDA to support TCP/IP
- Proliferation of DRDA implementations
- Discussion of extended data access to heterogeneous data

Relational Database Resource Manager

This paper describes the role of the Relational Database resource manager in the Open Blueprint structure:

- The overall function it provides
- The APIs it supports
- The open protocol it supports between its client and server components
- Its relationship to other components in the Open Blueprint

The Relational Database resource manager is a component of the overall Open Blueprint, and provides relational database facilities to applications and resource managers in IBM and non-IBM environments through a standard application programming interface (API).

The Relational Database resource manager can be implemented by one or more products (or sets of products) if these products conform to the definition included here. This paper includes examples of products that provide the component being described.

Relational Database Resource Manager Function and APIs

The Relational Database resource manager provides robust relational database services and protocols that operate against databases stored anywhere in the network. For many applications this ability provides sufficient distributed support so that the application can be written in a non-distributed style.

The database facilities are divided into basic groups that are targeted for application access to data through Structured Query Language (SQL). These groups are: application programming (including database BIND) and replication of relational data. Each of these groups is discussed in this document.

SQL - The Relational Database Access Language

The APIs supported by the Relational Database resource manager rely on the grammar defined by SQL, which provides powerful non-procedural operations for accessing and managing data contained in relational databases. In a relational database, data is conceptually organized in tables and one table is related to another by data values, not linked by records. SQL provides the capability to retrieve, insert, update and delete data. Elements of the language deal specifically with relational constructs such as tables, rows, and columns, and with operations such as the join and union of multiple sets of data. The most commonly used relational operations are set-oriented and perform actions on data items identified by predicate specifications. In this example, the WHERE clause contains the predicate specification that will produce a list of people's names who might be eligible to retire.

```
SELECT    Name
FROM      Personnel.Jacket
WHERE     Age => 55
```

Similar predicate-driven specifications allow UPDATE and DELETE commands to operate on sets of data. INSERTs occur one at a time, but they can be grouped with application control into a single unit of work.

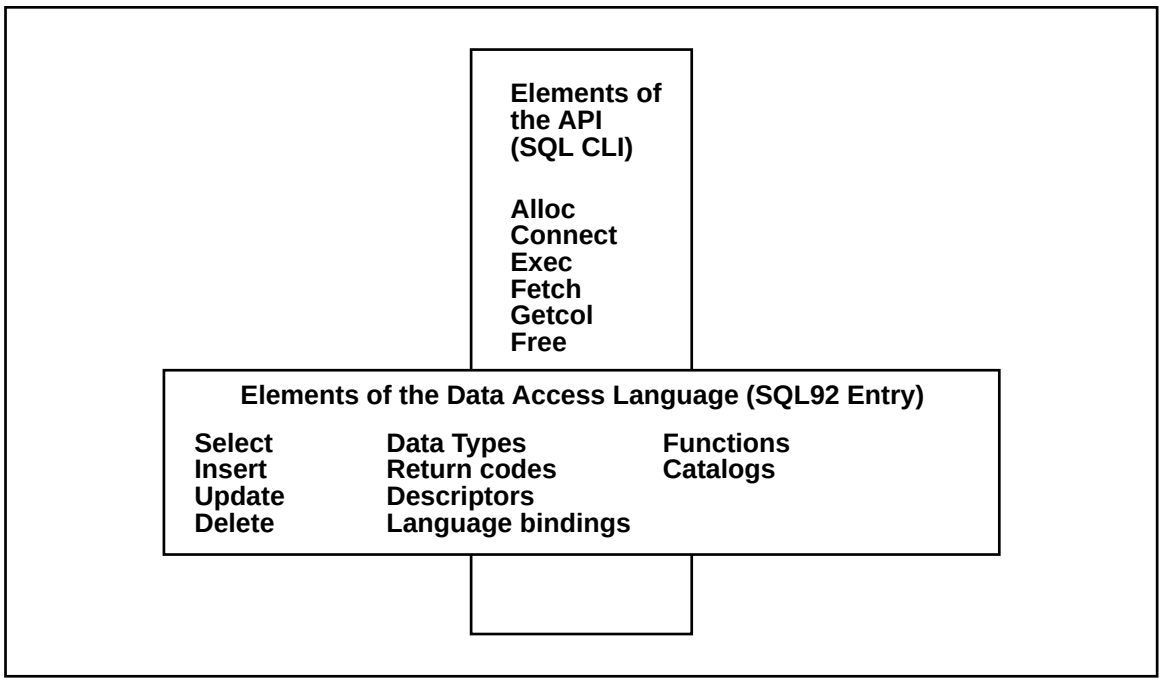


Figure 1. Language and API Standards

All SQL operations are performed under access controls and within the context of a unit of work. If an operation is not allowed for a particular user, it is rejected. The granularity of this access control is as fine as administrators want it to be, including limiting access to particular columns (fields) or rows (records) of a table. For any unit of work, the application can request backout to undo operations that produced undesirable results. Only when the application says things are OK are any changes committed and made visible to others in the system.

Recognized standards organizations such as the International Organization for Standardization (ISO), the American National Standards Institute (ANSI), and the industry consortium X/Open have formal definitions for the SQL language. The result is that a very significant amount of SQL has become standard in the industry, and portability of applications between different data management systems is becoming more and more achievable. The current definition of SQL from ISO and ANSI is popularly called SQL92, which includes three levels of conformance: entry, intermediate, and full. At this time, SQL implementations are expected to conform to the entry level. Conformance to the remaining levels will be expected at some (as yet unspecified) time in the future. DB2 will be certified as conforming to entry level SQL92.

Currently, there are two standard APIs that use SQL to access data in the database, both of which are supported by the Relational Database resource manager:

- Embedded SQL
- Callable SQL

Embedded SQL: In embedded SQL, the SQL statements are interspersed directly into the application program in sequence with the procedural language statements. Programming languages typically do not understand SQL statements, so something must be done to make the SQL statements acceptable to a programming language compiler. Embedded SQL is typically converted by a language preprocessor into a series of assignment statements and procedure calls that are compatible with the programming language and are then compiled along with the rest of the application program statements.

Program preparation[1] is the process used to preprocess SQL statements, compile the application program, and bind program variables and parameters to the target data management system.

BIND operation: Two fundamental styles can be used to write applications to access SQL relational databases: *static SQL* and *dynamic SQL*. With dynamic SQL, the entire SQL statement to be executed is passed to the database manager for evaluation. With static SQL, which is part of the embedded SQL style, the statement is given to the database manager ahead of time during a process known as *bind*.

When static SQL is used, it is common that values of some variables will not be known until the application is run. To allow this flexibility, references to variables in the application program are used.

The ability to bind an SQL application is an IBM defined extension to the ANSI SQL standard that provides compiled level of performance for SQL statements. Variables in the application programming language can be referenced by the SQL statements if they are bound. When binding occurs, an optimizer looks at the statement and determines the best method for performing the SQL operation. The result of the bind is a compiled object called a *package*[2] that is maintained by the DBMS and is executed later during runtime. By pre-selecting an optimized path through the database and eliminating the need to interpret the SQL statements during runtime, this bind process enables the development of performance-critical applications.

Binding also gives a level of indirection for access control. The user who binds the package (binder) must have the authority to perform whatever operations are called for by the SQL statements being bound. (For example, if update operations are in the package, the binder must have update authority to update the target tables.) The binder then has the authority to allow (GRANT) others to use the package. Users do not need authority to update the data directly, only to run the package. Thus, many people can be authorized to update the master database, but only using pre-defined packages. The programs can be certified as trustworthy before being put on the systems. Because users can only update data using authorized programs, they are unable to damage the database.

Applications using embedded SQL are supported on all the platforms that support DB2 applications today:

- OS/2
- Windows (3.1, 95 and NT)
- DOS
- AIX
- HP-UX
- Sun Solaris
- Sinix
- Santa Cruz Operations (SCO)
- OS/400
- MVS/ESA
- VM/ESA
- VSE/ESA

Callable SQL: Callable SQL is an alternative API for using SQL with a DBMS. In contrast to embedded SQL, the SQL statements are not embedded within the application as executable statements, but instead are passed as character strings through function calls to the DBMS. These function calls provide the same capabilities as embedded SQL. Using a series of function calls, an application submits an SQL statement for processing, retrieves the resulting data (if applicable), and inspects status information.

Callable SQL does not use a program preparation process, so callable SQL is always *dynamic*. Functions built into callable SQL replace the processing achieved during program preparation. Because the application program is not preprocessed, the application can be independent of the DBMS. However, callable SQL statements do not have the performance advantage of pre-processed embedded SQL statements, which have had an optimized runtime path predetermined for them.

Application developers who are interested in accessing multiple database management systems with their applications find callable SQL more flexible. Callable SQL functions allow the application to identify which database management systems are available in the runtime environment and to determine some of the

information about each. This information includes data server names, server product identification, and SQL dialect conformance. Being able to determine this information at runtime means the application can be adaptable to the capabilities of the data source.

X/Open, ANSI, and ISO are working on the standard specification for SQL. This standard specification, the SQL Call Level Interface (SQL CLI), is expected to be the standard base for future implementations of callable SQL.

Applications using SQL CLI are currently supported on OS/400 and all platforms for which there is a DB2 client application enabling function, currently:

- Windows (3.1, 95, and NT)
- OS/2
- AIX
- HP-UX
- Sun Solaris
- Sinix
- SCO

IBM plans to provide support for SQL CLI in DB2 MVS.

Open Data Base Connectivity (ODBC): Although X/Open SQL CLI provides a call/return API to invoke SQL functions, a vendor-specific database implementation might be limited to accessing only that vendor's particular DBMS. In addition, applications often need to access multiple DBMSs in the same application using a single, consistent SQL CLI. To support access to multiple DBMSs, there is an architecture that supports a single API and multiple back-end adapters or drivers, one for each type of database management system. (See Figure 2 on page 7.)

This architecture defines a driver manager layer which handles the CLI and routes SQL requests to the data access drivers. The interface that defines how the DBMS-specific drivers or back-end adapters plug into the driver manager layer is referred to as a service provider interface (SPI).

One implementation of this architecture is ODBC.[3] Since its early implementations, ODBC has been delivered by software vendors on several platforms. ODBC can access several back-end data stores, including

- DB2 for OS/2
- DB2 for AIX
- Oracle
- Sybase
- FoxPro
- Excel files
- Flat files

The Relational Database resource manager supports embedded SQL, callable SQL, and ODBC APIs and the drivers that direct requests to DB2 servers.

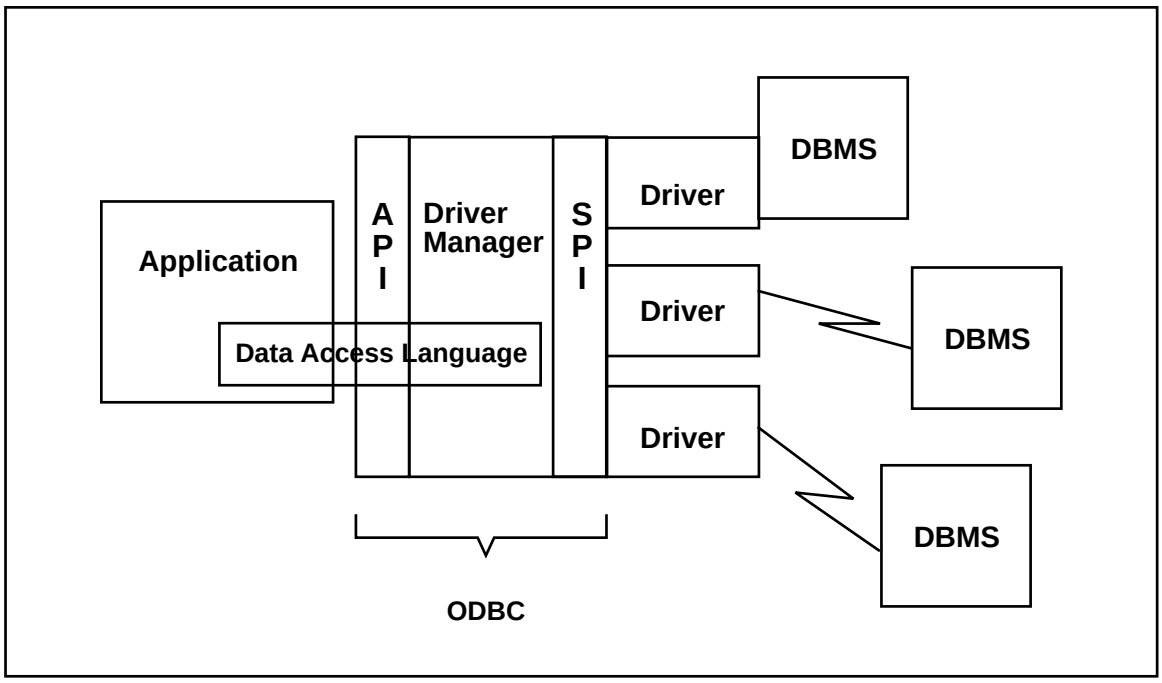


Figure 2. ODBC Architecture

Java Data Base Connectivity (JDBC): JDBC is an emerging industry interface that is used by Web-based Java applications for database access. It is similar in structure to ODBC and is provided on any platform that supports the Java class libraries and application environment. For more detail on JDBC, refer to “Access to the Database from Java Applets or Applications” on page 22.

Relational Database Support for Advanced Data Types: Several functional extensions are being added to ISO SQL and to IBM's relational databases to provide easier and more efficient support for objects and other unstructured types of data. These extensions provide a better match (more precise mapping) between an object and its stored representation. Within these extensions, user-defined functions and user-defined data types are being defined for inclusion in future versions of the standard. IBM is adding support for binary large objects (BLOBs) to IBM's relational databases to support objects in general, specific applications, and Open Blueprint resource managers such as Multimedia and Digital Library.

Functional extensions are described in the next several sections.

User Defined Functions: User defined functions allow encapsulation of data and methods to produce desired results. The user can write the function to access data from the database, perform calculations and aggregations on the data, and return a scalar result. A simple example of this is calculating a person's age from a date-of-birth column and today's date.

User Defined Data Types: User-defined data types provide strong data typing between stored data and user-written functions. The data typing is enforced by the database engine. Strong data typing reduces the risk of logical errors in an application. For example, a trigonometric function written to accept arguments expressed in degrees could not be invoked using arguments defined as radians, where degrees and radians are user-defined data types. Conversely, if there are two equivalent trigonometric functions, one for degrees and one for radians, the database manager ensures the selection of the correct one.

BLOB (Binary Large Objects) Storage and Retrieval: BLOBs are strings of non-formatted data that are used to store objects when the size of the object is larger than the normal maximum size of a base data type. The BLOB is stored as a user-defined data type derived from a character string base data type. Examples of BLOBs are multimedia data types such as audio, image, and video. In the cases where a BLOB is too large to store within the database, or when it would be inefficient to do so, the BLOB is represented in the database by a unique object identifier and the actual BLOB is stored in a separate data store (for example, a File resource manager video server).

Additional functions are provided in the BLOB support to allow retrieval and manipulation of parts of a BLOB in a manner similar to substring and concatenation operations.

Extended Data Access

Some implementations of the Relational Database resource manager allow access to heterogeneous data from different sources with a single SQL statement and a single interface, giving the application the image of a single database. Thus, a single query can access multiple data sources, find the data, and join it together before presenting it to the user or application. The data sources can be relational or non-relational, local or remote.

The single API can be either the embedded form of SQL or the callable SQL (either SQL CLI or ODBC). No matter what the source is, the application only has to use this single API. The resource manager function will map it as required to the other APIs that might be required for a particular data source. The data locations, SQL dialects, networking protocols, operating systems, data types, error codes, and functional differences are transparent to the requesting application.

An example of support for the extended data access function is IBM DataJoiner. In addition to being able to access databases managed by the IBM DB2 family, DataJoiner accesses source data from Oracle, Sybase, Informix, Microsoft's SQL server, IBM IMS, IBM VSAM, and more than 60 other sources through third party gateways.

Relational Database Resource Manager as a Distributed Resource Manager

Logical Structure

The logical structure for database usage in a distributed environment is illustrated in Figure 3 on page 9. It consists of applications, database management systems, and a collection of client and server components.

The communication between the client and server components in the Open Blueprint is defined by Distributed Relational Database Architecture (DRDA), which consists of a set of protocols that describe the commands, responses, data formats, and overall process flow. The client and server components of the Relational Database resource manager are referred to in DRDA as the Application Requestor (AR) and Application Server (AS) components, respectively. Because of the openness of DRDA, either or both components could be provided by a given software vendor and each would be able to talk to its counterpart provided by any other software vendor. For example, many non-IBM database and communications software vendors currently provide DRDA Application Requestors (client components) that talk to IBM DB2 family of Servers.

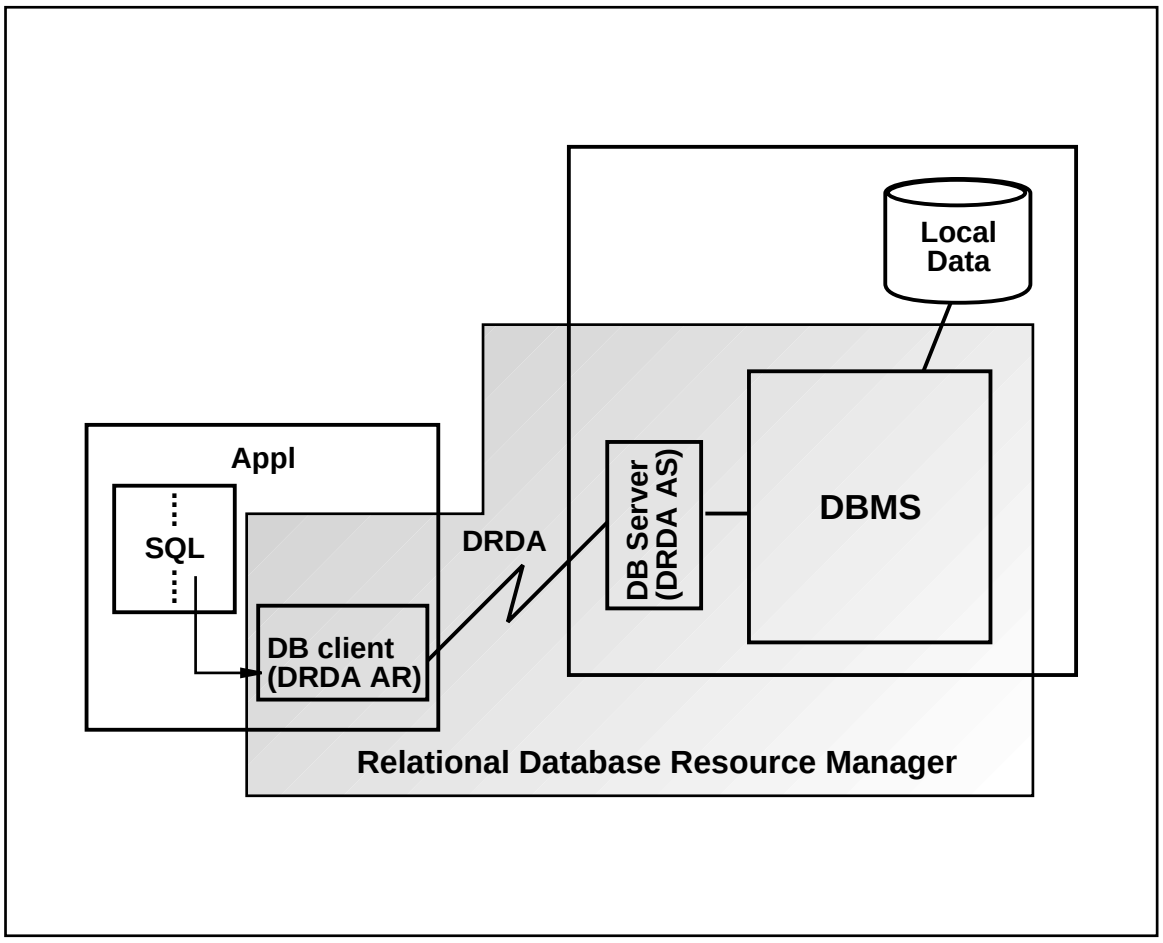


Figure 3. DB Clients and Servers—Logical Relationships

Applications

The applications in Figure 3 have a union of facilities at their disposal. They can be written in any of the programming languages supported in the *application's* environment and can use any of the services normally provided in that environment. In addition, they can request Database Management System (DBMS) services provided by any of the DBMSs in the network and take advantage of the ***full*** functionality of each. It is actually possible to deploy an application in a distributed environment that could not operate as a single local operation in any environment of the network. This is because it might use DBMS-specific functions of multiple DBMSs within that single application. This support for the union of facilities allows for easy exploitation of new facilities as they are introduced into a network while continuing to support existing systems and applications without change.

Relational Database Client

The Relational Database resource manager client is a combination of the component that is invoked to process the SQL API (for example, the Client Application Enabler, or CAE, component of DB2) and the Application Requestor (AR) component in DRDA. The client provides the services required to place SQL and BIND requests on the network to route them to the appropriate DBMS, flowing these SQL requests as DRDA function protocols. Besides building the DRDA flows, the database client is also responsible for such things as providing appropriate security credentials with which the user authentication can be performed at the server. When the response is received, the database client provides the answer set and status information to the application in the expected format.

Relational Database Server

The Relational Database resource manager server is a combination of the Application Server (AS) component of DRDA and the actual DBMS database engine. The server receives the request from the client, provides it to the DBMS for processing, then returns the response to the client. The server is also responsible for ensuring that the user requesting database services is authenticated, providing access control for the objects it manages, registering with and responding to a local transaction manager, notifying the database client of any server errors, and performing other environmental-type functions.

The server component of DRDA, the AS, is so closely tied to the actual DBMS, that they are often thought to be the same component. However, it is a logically separate component that can be physically packaged separately from or included with the DBMS. All of the DB2 servers and IBM DataJoiner (see “Extended Data Access” on page 8) contain the DRDA AS function and can be accessed by any client that "speaks" DRDA.

The Relational Database Resource Manager's Open Protocol

DRDA is the distributed relational database protocol that is defined as part of the Open Blueprint and supported by the IBM DB2 family of products.

Distributed Relational Database Architecture (DRDA)

DRDA is an open, published architecture that enables communication between applications and database systems on disparate platforms, whether those applications and database systems are provided by the same or different vendors and whether the platforms are the same or different hardware/software architectures. It is implemented by IBM database products and by other key non-IBM database and communication software providers (see Figure 6 on page 14). For detailed information about the DRDA architecture, see Appendix B, “Bibliography” on page 27.

DRDA is the basis and coordinating architecture for distributed database among IBM platforms and any multi-vendor platforms on which DRDA is implemented. The architecture specifies SQL-related rules and processes that are required to BIND SQL statements at remote DBMSs and to execute those statements and others at run time.[4]

DRDA is insensitive to the level or dialect of the SQL used by the application and will flow any SQL request to the server. One advantage of being independent of SQL syntax is that a DBMS can be introduced into the network and the new DBMS can be accessed immediately by an application on any client without requiring all clients or servers in the network to be upgraded.

In addition to supporting the SQL data manipulation commands, DRDA also flows the administrative commands such as GRANT, REVOKE, CREATE, ALTER, and DELETE, so the database can be managed from a remote client.

DRDA defines the format in which the user data must be specified. DRDA is built on several underlying architectures which are shown in Figure 4 on page 11 and described below.

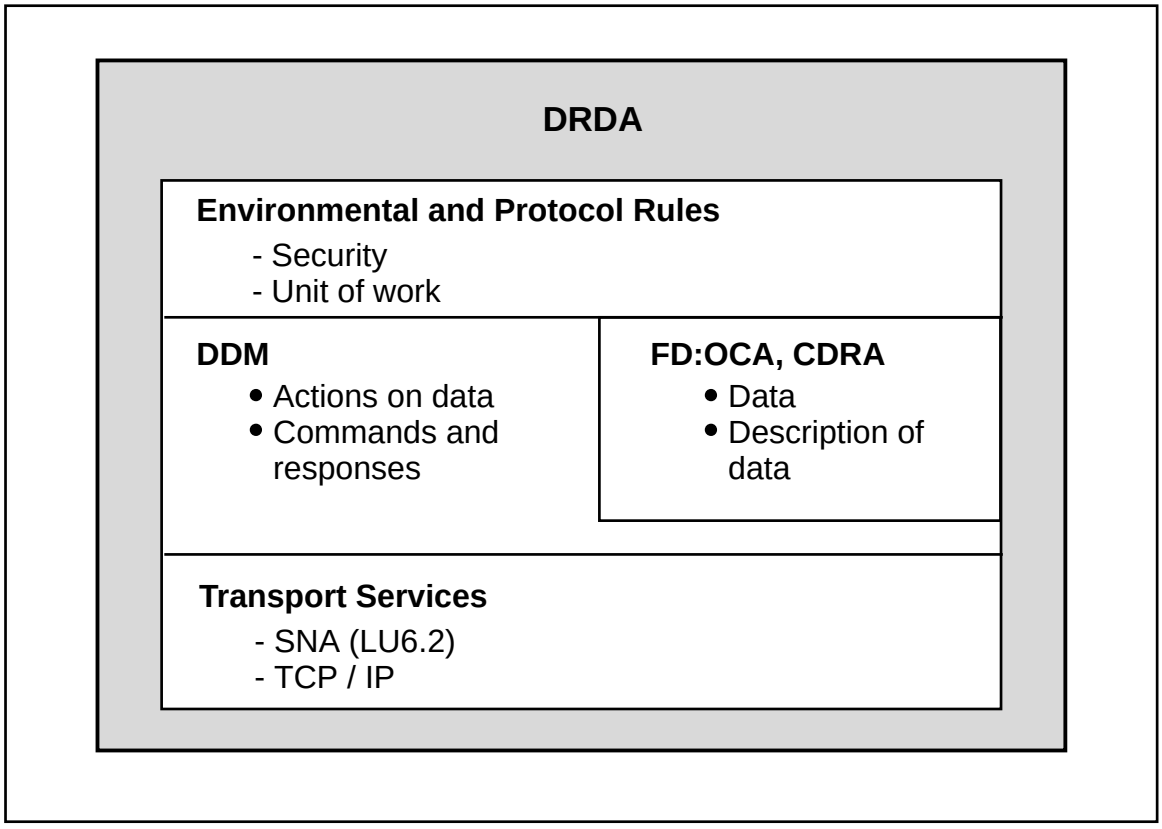


Figure 4. DRDA Architecture Composition

DDM The Distributed Data Management (DDM) architecture provides the overall command and reply structure used by the distributed database. Fewer than 20 commands are required to implement all of the distributed database functions for communication between the Application Requestor (client) and the Application Server.

FD:OCA The Formatted Data Object Content Architecture (FD:OCA) provides the data definition architectural base for DRDA. Descriptors defined by DRDA provide layout and data type information for all the information routinely exchanged between the Application Requestors and servers. A descriptor organization is defined by DRDA to allow dynamic definition of user data that flows as part of command or reply data. DRDA also specifies that the descriptors only have to flow once per answer set, regardless of the number of rows actually returned, thus minimizing data traffic on the wire.

CDRA The Character Data Representation Architecture (CDRA) provides the character integrity mechanisms for DRDA. All character data is tagged with Coded Character Set IDentifiers (CCSID) which specify the character set, code page, and encoding scheme employed by the tagged character strings. This tagging ensures unambiguous understanding of the bits in the string so that the meanings of the characters are preserved as they are processed. For example, “A” stays “A” and “<” stays “<.”[5]

By using CDRA, DRDA also ensures round-trip integrity between any requestor/server pair where conversion does occur.

LU 6.2 SNA's logical unit type 6.2 (LU6.2) provides the architectural model for the conversational-style communications used by DRDA. LU 6.2 also provides the architectural model for the unit of work support used by DRDA to coordinate updates across multiple sites.

When using SNA LU6.2 as the communications transport, DRDA also uses the security, unit of work, and session outage notification functions provided by this protocol.

TCP/IP DRDA can also flow over TCP/IP networks. In this case, it follows a model parallel to that for LU6.2 but uses transport-independent mechanisms for the capabilities that were provided as part of LU6.2 such as security, unit of work, and two-phase commit support.

DRDA Configurations

Access to DRDA-enabled servers is supported in different physical configurations, as illustrated in Figure 5 on page 13. These are referred to as:

- **Direct connectivity** from the client to the server
- **LAN Server connectivity**, which supports several clients and provides the actual DRDA communication to the server from a gateway machine
- **Middleware connectivity**, whereby the middleware product provides access to several heterogeneous data stores using their proprietary APIs and providing access to all DRDA-enabled servers. An example of this type of connectivity is IBM DataJoiner (see “Extended Data Access” on page 8).

The IBM product that provides DRDA client connectivity from workstation clients is Distributed Database Connection Services (DDCS); it supports applications using embedded SQL, SQL CLI, and ODBC. It can be configured either as a single-user version, where the DRDA client component exists right at the client machine, or as a multi-user version on a LAN server. These would be examples of direct connection and LAN server connection, respectively, and can connect the client to any of the DB2 servers or to any DRDA-enabled server.

Depending on the specific vendor, the non-IBM implementations of DRDA support one or more of the above configurations.

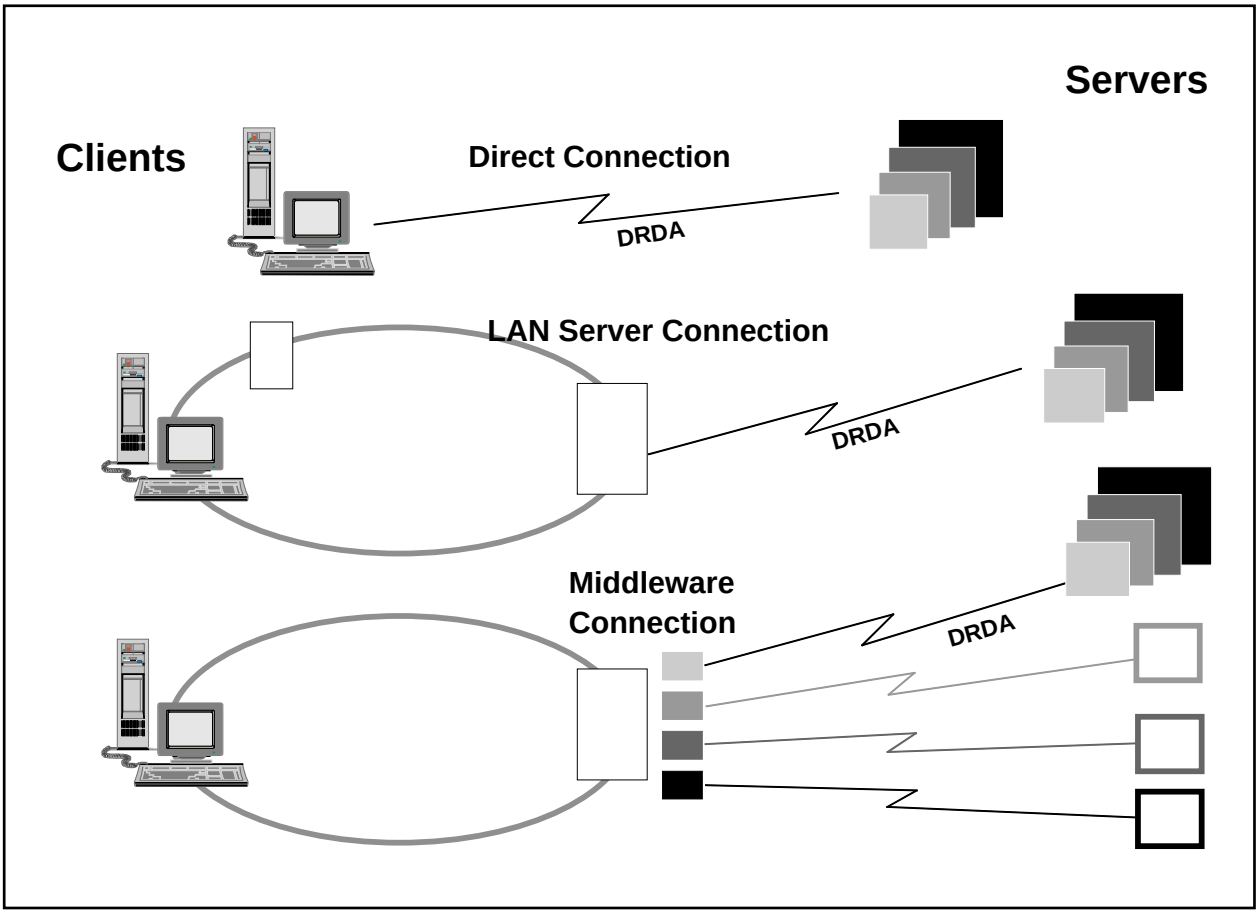


Figure 5. DRDA Configurations

Deployment of DRDA

DRDA has been implemented by many software vendors in addition to IBM on many different platforms. Its client function exists on all key platforms and it can access several different data stores in addition to DB2 (both relational and non-relational). Figure 6 on page 14 illustrates the many platforms for which there are DRDA client or server functions provided, in both direct connect and LAN Server connect configurations.

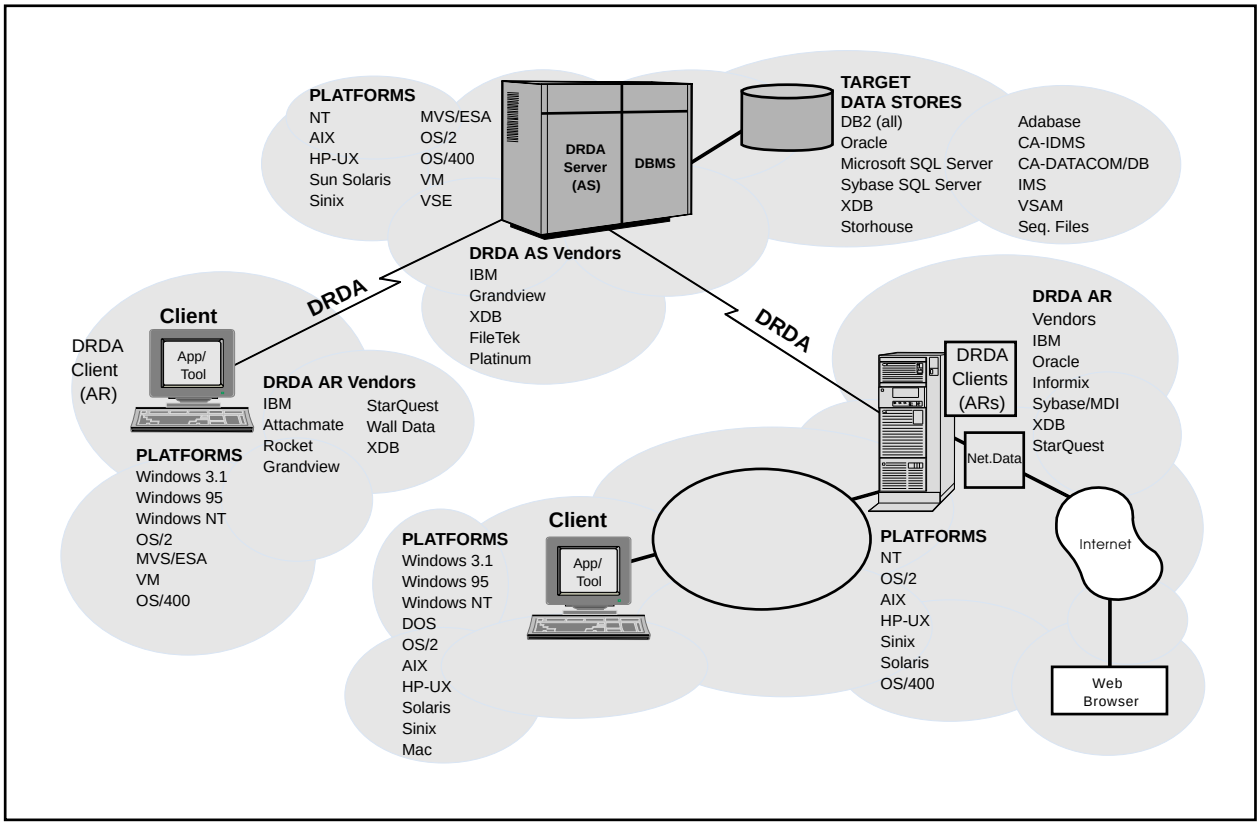


Figure 6. DRDA Deployment

Given that DRDA has been designed to support a broad range of database interoperability requirements, and because it has been implemented by several software vendors on all the key platforms, IBM has offered DRDA to The Open Group (formerly the X/Open Company and Open Software Foundation) for inclusion in its Common Applications Environment (CAE).

Database Administration

Replication

There is a requirement to extract subsets of data from a relational database and to distribute and maintain copies of that data at several places in the network. By using copies of data, performance of operations against the data can be improved by reducing or eliminating network delays and by reducing contention on the data. By automating the update of these copies, the appropriate data can be provided for any application at the right place in the network.

Creating copies of tables has an additional benefit in that it allows an installation to distinguish between the operational usage of the data (updating) and informational usage (browsing) and it keeps the updaters and the browsers out of each other's way. Locks held by updaters (exclusive locks on a small amount of data) prevent any browsers from operating. Conversely, locks held by browsers (shared locks on large amounts of data) prevent any updaters from operating. Therefore, updaters and browsers can co-exist best by operating on separate copies of the tables.

The replication function includes automatic facilities that keep copies up to date (to the specified level of currency) without intervention by operators or users. These automatic facilities also ensure that the actual replication activity is not done at prime time regardless of time zone and peak load hours. This replication

is accomplished by using distribution facilities that completely de-couple the updates to the operational data from their application to the target tables.

Assuming differential updates are possible, the system which has the source data being copied creates a small file of changes which are subsequently distributed to the DBMSs where the copies reside. This can be done at a convenient time for the base data system and at time intervals that can range from every minute up to hours, days or weeks.[6]

The Relational Database resource manager data replication is delivered by products such as DataPropagator Relational and DataPropagator Non-Relational. With these tools, data can be copied to and from DB2 databases, either by replicating only the changed data or by doing a total refresh of the source data.[7]

Replication environments can be extended by using IBM DataJoiner with Data Propagation products. By incorporating DataJoiner, data can be replicated to and from the many data sources supported by DataJoiner, as described in “Extended Data Access” on page 8. DataJoiner provides access to current and point-in-time data at diverse, remote data sources.

To support data sources and targets in addition to those addressed by DataJoiner and the replication solution products, IBM has defined a data staging area. The staging area enables non-IBM applications to extract data from a foreign source and place it in the staging area. From there it is subsequently copied to specified targets by a product such as Data Propagator Relational. Similarly, an independently written copy (or “apply”) program could be used to take data out of the staging area and copy it to one or more targets. The result is an open architecture for data replication.

For more information on IBM replication and the staging area, see Appendix B, “Bibliography” on page 27.

Systems Management

The Relational Database resource manager has multiple relationships with systems management. It ***is*** a managed resource (a resource manager), it ***manages*** resources (tables, DASD, and user access lists), and it provides database services that are ***used*** by the managing applications.

As a managed resource, the Relational Database resource manager responds to the same requests as any other resource manager. It can be created, deleted, configured, started, stopped, or monitored for status and performance information. Activities like this are usually carried out by a Database Administrator, who uses database management tools that interface with the database through SQL or by using database-specific utilities, such as reorg or backup. In their role as managed resources, the DBMSs are managed by IBM's DataHub products, DataHub for UNIX and DataHub for OS/2. Both provide a consistent view of the database world to database administrators. DataHub for OS/2 manages DB2 Servers on MVS, OS/2, AIX and VM from an OS/2 control point. DataHub for UNIX provides comprehensive database administration and monitoring based on business policies for DB2, Oracle, Sybase and Ingres on AIX, HP-UX and Solaris platforms from an AIX, HP-UX or Solaris control point. As a first step to integrating database and systems management with network management, DataHub also interoperates with network management applications through Simple Network Management Protocol (SNMP) alerts.

As a managing resource manager, the Relational Database resource manager manages such resources as tablespaces, tables, and indices; enforces business policies defined on the database objects; ensures integrity, security and availability; and provides status and performance information about the resources when queried. It responds with service information (for example, alerts and first failure data capture) and performance information (for example, counters relating to work performed over some interval). For

example, if a status request is received with a table as the object of the request, the DBMS responds with meaningful information such as “online,” “idle,” and “undergoing repair.”

Relational Database resource manager system management support will use the Open Blueprint Systems Management Services and Framework APIs defined as part of the Tivoli Management Environment (TME 10). This will result in integration with existing solutions from other software vendors for heterogeneous database systems management, including Informix, Oracle, and Sybase.

As a provider of services for systems management applications, the DBMS provides the underlying data store services and access to those services through the SQL API.

The Relational Database resource manager also includes an SNMP agent that supports the standard RDBMS Management Information Block (MIB) as defined by the Internet Engineering Task Force (IETF).

Relationship to Other Resource Managers and Services

In the overall flow of an application that uses the Relational Database resource manager, the application interacts with the database through a series of SQL calls as part of its application logic. When the application initially connects to the database (either implicitly or explicitly), a number of other things happen, many of which involve interaction with other resource managers in the Open Blueprint.

For example, after reading the Directory entry to find out how and where to access the server for the specified database, the database client obtains credentials by which the user can be authenticated at the server. The client also begins a unit of work for that application.[8] This unit of work remains in effect until the application issues a COMMIT or ROLLBACK command or until it terminates.

The following sections describe the relationships between the Relational Database resource manager and other resource managers and services in IBM's Open Blueprint in more detail.

Identification and Authentication Resource Manager

The Relational Database resource manager depends on having authenticated identifiers for all users of the system. For local users, this information is requested when the user initiates contact with the database. It is assumed that the user was already identified and authenticated before execution of the database request began. Therefore, the database is just finding out system information that is already known and associated with the executing process.

A similar process is followed for users whose requests arrive at the DBMS over a network (from another system). In the context of IBM's Open Blueprint, it is assumed that the user has already been identified to and authenticated by the Identification and Authentication (I&A) resource manager. When the application being run by the user makes a request to access a database (that is, through an SQL CONNECT statement), the database client code passes the Identification and Authentication resource manager the database server name and the previously obtained user authentication information. The Identification and Authentication resource manager then returns an encrypted token or “ticket” that is sent to the Database Server with the remote request. A DDM construct is used by DRDA to carry this ticket to the remote database where it is extracted from the datastream, parsed, and authenticated at the remote server.

The API that is used to invoke the Identification and Authentication resource manager is the Generic Security System (GSS) API. Because this API is used as delivered by Open Systems Foundation (OSF) Distributed Computing Environment (DCE), the distributed database's use of DCE security is not dependent on an IBM DCE implementation. This capability is supported on any platform where the IBM DB Client Application Enabler (CAE) and a DCE implementation are supported.

In some cases, DCE security services must co-exist with platform-specific security services (such as RACF on MVS), in which case authentication of a given user can require coordination between the two. In such cases, it is assumed that the database server code can obtain a local platform-specific id from the Identification and Authentication resource manager at the server, which is derived from the DCE principal ID included in the ticket. For more details on the specific technique used to accomplish this, refer to the *Open Blueprint Security Resource Manager* component description paper.

If the underlying communications services being used to transport the database requests to the server have their own built-in security services (for example, flowing user id and password in LU6.2), these are independent of the use of DCE tickets for authentication. In fact, if both the underlying communication service's authentication function and DCE (Kerberos) Security Services are used, the ticket included in the datastream will take precedence.

Access Control: Access control for database objects is performed by the DBMSs themselves, by implementation of CREATE, ALTER, DROP, GRANT and REVOKE SQL statements, which are part of the ISO and ANSI standards definitions. Objects on which access is controlled and the privileges allowed vary from system to system, though there is a set of objects and privileges that are common. For example, the SELECT privilege against TABLE objects is common to the relational databases on all the IBM platforms. The current design for IBM's Open Blueprint relational database is for the resource manager to provide access control over its own resources (tables, views), based on the end user having been previously authorized to use them through the SQL GRANT (or REVOKE) command.

For users whose work arrives at the DBMS over a network (from another system), the DBMS extracts user identification information from the communication flows or from the DRDA Security token construct. This user information is used for authorization to resources managed by the DBMS. However, the SQL language defines the format of the user ID to whom privileges are granted differently from the globally unique principal ID as provided by the Open Blueprint. Therefore, the database server code maps the global principal ID to its local format, in conjunction with the Identity Mapping and Credential Transformation resource manager.

Communication

Communication between the database client and server code is performed using DRDA and the conversational paradigm as provided by the Conversational resource manager. These conversations are used for application preparation (BIND processing) and application execution (execute SQL processing).

The Relational Database resource manager uses the Open Blueprint Conversational Communication model between its client and server components. Although this is an SNA or OSI model, the distributed database flows can also be carried over TCP/IP networks using either the Common Transport Semantics, as provided by the AnyNet family of products, or by flowing over native TCP/IP using a Sockets communication interface. The Relational Database resource manager assumes the interface to Open Blueprint Common Transport Semantics is included in the function provided underneath the Conversational resource manager API.

While DRDA was originally designed to use LU6.2 for communication between its client/requestor and server components as well as being the base for unit of work support, DRDA has recently been extended to support TCP/IP as an underlying transport mechanism. To do this, DRDA has been changed to support such functions as unit of work support, session outage notification, and security in ways that are not dependent on the underlying communications service. These functions are provided either through new DRDA constructs by themselves (for example, for unit of work support) or through new constructs in conjunction with other resource managers (such as Security).

Additionally, there is communication between database clients and servers on a LAN that use native transports and do not support DRDA. In these cases, other resource managers such as Directory

Services or Identification and Authentication are also used. However, because the function protocols used between the client and server are not open protocols such as DRDA, the same level of interoperability between clients and servers cannot be achieved. In these cases, a gateway is provided at a server that speaks DRDA to the rest of the network (see “DRDA Configurations” on page 12).

Transaction Management

Database Management Systems (DBMSs) provide services to create, destroy, insert, update, or delete data that operate in the context of a logical unit of work (LUW). This work is performed within a commit scope, which is terminated by the SQL COMMIT or ROLLBACK commands.[9] When a commit is performed, all the DBMS work requested by the application is “hardened” in the database. When a rollback is performed, all tentative changes to the database are removed.

The Relational Database resource manager associates a logical unit of work identifier (LUWID) with the user's work as soon as the user makes a request that requires one. When an application issues a CONNECT command, the Relational Database resource manager obtains the LUWID from existing system information, if the LUWID exists. If an LUWID does not exist when the CONNECT is issued, one is created by the database client in the format specified by DRDA.

When the Relational Database resource manager operates under the Transaction Manager resource manager, database resources can be part of a broader unit of work involving other resources. In this case, the Relational Database resource manager responds to calls from the Transaction Manager resource manager.[10] The original request for commit (or rollback), however, originates from the application using the Transaction Manager API, for example, X/Open's TX API. Or, if only relational database resources are involved, the Transaction Manager resource manager invoked by the SQL COMMIT or ROLLBACK commands and the Relational Database resource manager can act as the syncpoint coordinator for the transaction. (For object-oriented applications the Object Management Group (OMG) Object Transaction Services (OTS) are used for managing transactions in that environment.)

Directory

Naming: Databases and objects within a database are named. Both these names are fixed in structure by SQL standards. Objects within a database have two-part names that are guaranteed to be unique by the DBMS. These names are maintained in catalogs or system dictionaries and are accessible to the processes that perform the SQL operations.

Database and database server names are guaranteed to be unique by inclusion in the Open Blueprint Directory name space (for example, by their position in a given cell in the DCE Directory name space).

The Relational Database resource manager understands and provides support for the lower two levels of database object names for the resources it manages, such as Tables. Such DBMS-managed objects are managed using techniques such as private database files or catalogs. Only the Relational Database resource manager has knowledge of the resources it manages (such as tables, views); these objects are not registered in the directory service.

Directory Objects: One of the key objectives of using the directory to locate a given relational database is to allow the database to be moved from one location in the network to another transparently to the application and without having to modify side tables in all the client machines. The Relational Database resource manager uses directory services to locate a specified relational database (RDB), understand its characteristics and determine whether the client is capable of communicating directly with the server for that RDB. To do this, the following directory objects are defined: a *database object* (per RDB), a *database server object* and a *database gateway object*.

When a request is made to connect to a given RDB, its entry and its associated database server entry in the directory are read to determine its network address, the functional protocols it understands, and other information about the server. The database client uses an implementation-specific technique (such as a local catalog or a dynamic mapping done by the application) to map the RDB name known by the application to its directory name. In this way, the directory name associated with an RDB can be changed at any time without requiring changes to the actual application.

If a client making a request determines that it cannot communicate directly with the RDB server through any of its supported protocols, then it locates a database gateway object through the directory that is capable of speaking those protocols. One of the attributes of the database gateway object is the address to which the request should be sent. The database gateway object then maps the request to the protocols required and sends it to the specified database server. Each client is assigned a database gateway object, which is to be used for this purpose. The association of certain gateway objects with given clients also provides the ability to direct remote requests from a client to a given gateway for either load balancing or network configuration reasons.

Because DRDA defines different stages of implementation, it is likely that databases will exist in the network at various levels of the architecture at any given time. Therefore, the database object would also contain an indication of the DRDA level of support provided by the associated server, such as multi-site update support.

Collaboration Resource Manager

Among the resources that the Collaboration Resource Manager manages is the database of documents that are created and shared among the group of users who are doing the collaborating. This type of database differs from a relational database in that the collaboration document store is not a transactional database and it usually deals more with fields of unstructured data than a relational database has done traditionally. However, it is likely that an enterprise has data that is used in both their operational relational database and their collaboration document store.

To support the case where the data is needed in both types of databases, there are mechanisms provided to automatically get and put data between them, without explicit action from the end user or application developer. By configuring administrative information that maps the desired fields between the databases, the data is asynchronously replicated between the relational database and collaboration document store using the standard resource manager APIs.

An example of a product that performs asynchronous replication between the relational database and collaboration document store is the NotesPump facility of Lotus Notes. NotesPump can replicate data between a Lotus Notes database and the DB2 database and to non-IBM relational databases using the technique described above. NotesPump can also be used with the capture component of Data Propagator Relational, which replicates specified changed data from DB2 sources to Notes databases. DataPropagator Relational's changed data tables look like a DB2 database to NotesPump. (See Figure 7 on page 20.)

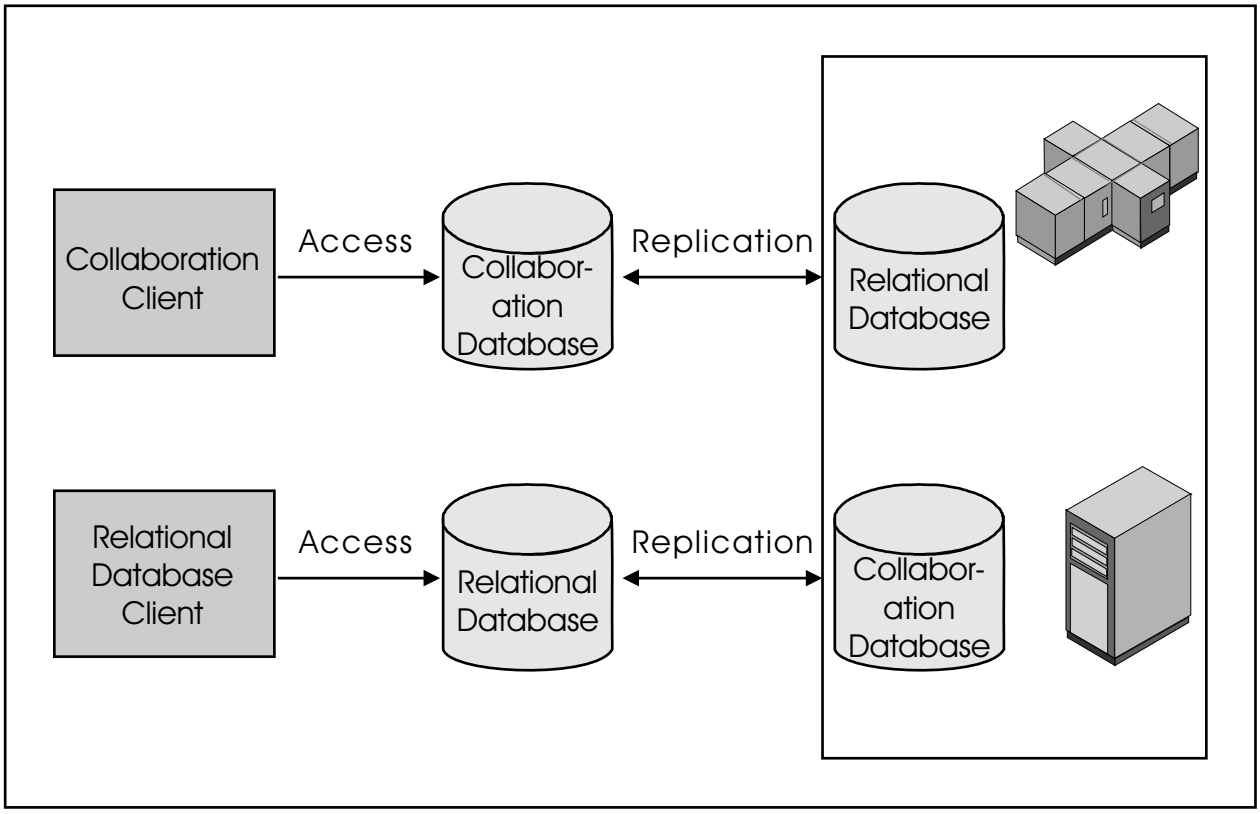


Figure 7. Collaboration and Relational Database Remote Managers

Relational Database and Persistent Objects

When application objects are to be permanently stored in some underlying data store, they are referred to as *persistent objects*. The services used to access persistent objects are referred to as *persistence services*, which are provided as a persistence framework. Because a relational database is only one of the underlying data stores in which such persistent objects might reside, persistence services in general will not be discussed in this paper.

Persistence services are based on the definition developed by the Object Management Group (OMG). In addition to this definition, IBM permits different types of persistent object access. These types range from a single level store, where the application just references the object to materialize it, to access at a much more granular level that is closely related to the underlying data store. A class library is available that relates directly to those available through SQL when the underlying database is a relational database.

For additional details on persistence services, including how an object in a relational database can be accessed using these services, refer to *Open Blueprint Persistence Services Resource Manager* component description paper.

Relational Database and Network Computing

One of the main ramifications of the rapid expansion of network computing and the Internet is to provide access to information and services to millions of new clients because of the use of Web browsers and HyperText Markup Language (HTML) pages. The Open Blueprint includes Web Browser and HTTP resource managers that support the associated standards for Web page content and the protocol used for transmitting them on the Internet, respectively.

These new technologies have brought increased accessibility to information in relational databases in different categories:

- Viewing formatted information (reports, ad so on)
- Inputting queries (or any SQL functions) from Web page GUIs
- Direct interaction with a database from programs invoked as Java applets or applications

Access to Standard Queries and Reports

Let's consider the first, and simplest, use first -- the ability to view formatted reports from any Web client. This especially relates to the Data Warehouse environment where an enterprise tends to have separate databases for its operational and informational data. To relieve the operational environment from the workload that might be placed on it from ad hoc queries used mostly for information access (that is, read only), many users will choose to replicate all or subsets of their operational data into "datamarts", or subject-area informational databases. This latter type of database is primarily accessed in two ways:

- Ad hoc queries that can then be modified to see different views of the data (what-if operations)
- Standard, or "canned" queries that produce a certain view of the data in the same way at any given time, such as sales results by region for a given time period.

The second method, views of data using standard queries, lends itself very well to the Internet and World Wide Web. The output of such queries (often produced by report generator tools used with databases) can now be formatted as HTML Web pages and accessed by anyone using a Web browser. The application developer (or tool writer) does not have to be sensitive to the device type on which the report will be viewed, because the Web browser will deal with that as long as the query output is formatted in HTML. End users needing the information do not have to worry about what device they are using as long as a Web browser is available that can access the report in which they are interested.

This type of access to data warehouse information on the Web is often referred to as *information publishing* -- the publishing of information as web pages instead of as the result of invoking specialized applications. Many Data Warehouse tool providers (query generators and so on) are now providing the ability to query results as Web pages and returning them to the screen interactively.

Using Network Clients and Web Browsers as Database Application GUIs

Before the advent of the Internet and Web browsers as methods for accessing all kinds of information, tool or application developers who accessed relational databases had to write GUIs that were sensitive to the device type being used at the client. By providing HTML pages that can be accessed by any Web browser, the developer is relieved of this responsibility and can reach end users that were never before accessible.

GUIs can be provided by normal HTML pages and forms, tailored to the application, and can be user friendly and similar to the interface used when viewing any other Web pages. Users are presented with a screen on which they are prompted to provide the values to be used for a SQL query (or update). Their input is then mapped to SQL-like HTML statements (users do not have to know anything about the SQL language). The user's input is sent to the Web server using the HTTP resource manager, where a CGI program is invoked that maps the HTML to SQL, then issues the request to the specified database.

An example of this type of Web access is IBM's Net.Data product. Using Net.Data, an end user can provide input to an application's GUI from any Web browser, which is sent to the server as HTML, then mapped to SQL at the Web server. Net.Data is supported on AIX, OS/2, MVS, AS/400 and Windows NT and can be invoked by any Web server that supports the CGI interface. The target database can be DB2 running on the same machine as the Web server and Net.Data, or the request can be directed to any DB2

servers using LAN or DRDA connectivity. If sent over DRDA, the SQL statements can be processed by any DRDA-enabled server. Net.Data connectivity is not limited to DB2 or DRDA-enabled servers, but can also be directed to any database for which there is an ODBC driver such as Oracle, Sybase, or Informix. Figure 8 on page 22 below, illustrates this type of connectivity, where Net.Data could be the application called through the CGI interface.

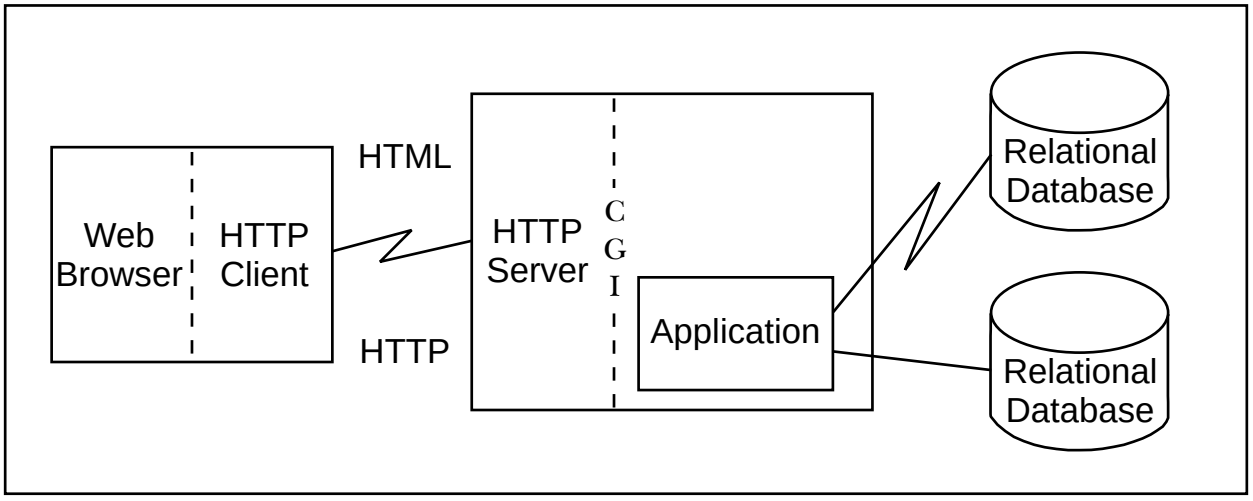


Figure 8. Network Access to Relational Database Data

Security: The Web Browser and HTTP resource managers can protect the data by using security functions that are transparent to other participants in this process. For example, secure connections between the Web client and server using protocols such as Secure Sockets Layer (SSL) or Secure HyperText Transfer Protocol (SHTTP) are supported and are totally transparent to the CGI gateway code (in this case, Net.Data).

Actual authorization to access data in the database is accomplished by providing a user ID and, optionally, a password. Assuming the user has been authorized to access the data using normal SQL GRANT commands, this ID and password are provided to Net.Data and used to access the database.

Access to the Database from Java Applets or Applications

The third method for using Web clients to access relational databases is through an interface provided by the Java programming language, as supported by the Virtual Machine resource manager. This interface, known as Java Database Connectivity (JDBC), is implemented as a Java class library used to access any database. Much like ODBC, it is used in other workstation environments. While JDBC is the interface used by a Java applet or application, it also defines the service provider interface (SPI) for database vendors (or anyone) to code drivers to that are invoked using JDBC to access their specific database. For example, a JDBC driver can be invoked that maps the JDBC call to SQL CLI calls that access DB2 servers.

A JDBC-ODBC bridge maps JDBC calls to ODBC, thus automatically enabling Java applications using JDBC to access any relational database for which there is an existing ODBC driver. While the JDBC-ODBC bridge is expected to be the primary method for Java applications to access relational databases initially, it is likely that many database vendors will provide their own JDBC drivers specifically optimized for their own database.

Relational Database and Mobile Computing

The Open Blueprint publication titled *Open Blueprint Support for Mobile Computing* describes the characteristics of mobile computing in general and the overall support for mobile computing in the Open Blueprint. So, those concepts will not be repeated in this paper. It is clear, however that mobile users need access to data that is stored on remote databases to which they are not constantly connected. In fact, it is also a requirement to be able to access this data (and perhaps update it), without being connected for a long period of time to reduce the connection costs involved, which can often be for long distance calls. The use of such data ranges from needing real-time access to the data to periodically refreshing the local data store so that local applications can be run without requiring a network connection.

The Relational Database resource manager support for the mobile user is based on providing the data that is required in the users' mobile (remote) environment when they need it, using the replication facilities of the Relational Database resource manager.

The technique that is used for mobile support is to replicate the data needed on the mobile workstation from the remote source. By having a local database on the mobile workstation, one can utilize all the techniques described in “Replication” on page 14 to replicate or refresh either a complete database or just specified changed data onto that workstation. So, replication in general addresses the need to supply the data to the mobile workstation by supporting the ability to replicate or refresh data (or both) locally either on demand or at scheduled times. The data requested can be either an exact copy of the database or just the changed data since the last replication. In both cases, the data can also be subset or filtered by specifying criteria (in the form of SQL statements) to select only certain rows or columns or both.

To minimize the connection time required to transfer the data and update the local database, one can specify that the replication be done in asynchronous mode. In this case, rather than downloading and locally applying each row or set of rows sequentially, all the requested data is "batch" downloaded and then applied locally after the session is terminated. This capability is also referred to as "mobile apply".

In another scenario, the data on the mobile workstation is part of a collaborative database. By using the techniques described in “Collaboration Resource Manager” on page 19, the data from a relational database can be replicated to the collaborative database on the mobile workstation. Using Lotus Notes as an example, data from DB2 servers can be replicated into the Notes database using the NotesPump function. This scenario can be extended to the Notes user in a mobile environment, then the data can be replicated from a relational database to the Notes database and the user can take advantage of all the disconnected support provided by Lotus Notes.

---

1 The program preparation process (including the BIND statement) has not been standardized, so is not part of the ANSI, ISO, or X/Open SQL definitions.

2 The results of BINDing are known by different names in different systems (such as access modules, plans) but in all cases the application executes this compiled object at run time.

3 ODBC is a database interface defined and popularized by Microsoft.

4 DRDA supports distribution of any SQL level or dialect, including extensions that are neither ANSI nor ISO standard. Therefore, references to the SQL interface imply references to *any* SQL facilities.

5 For both numeric and character data, represented by FD:OCA and CDRA, respectively, the DRDA rule for conversion is “receiver makes it right.” This ensures that at most one conversion is done for any data sent between requestor and server, maximizing data integrity and minimizing process overhead spent on unnecessary conversions in the case of like-to-like databases.

6 There can be several target sites for a single update. Each can be scheduled for processing at its own convenience.

7 IMS DB and VSAM are also supported as sources of data for replication.

8 Depending on what level of DRDA is supported, a unit of work can include updates at either a single or multiple remote sites.

9 COMMIT and ROLLBACK are part of SQL as defined by ISO.

10 The transaction management function can be provided either as part of the underlying operating system, such as MVS, or by a separately packaged component (usually part of a TP monitor), such as CICS, Encina or Tuxedo.

Appendix A. IBM Relational Database Resource Manager Client and Server Platforms

This section describes the platforms and configurations supported by IBM implementations of the Relational Database resource manager

The Relational Database resource manager can be implemented by combinations of products. For example, the IBM products that implement a Relational Database resource manager are: the DB2 servers, the DB2 Client Application Enabler (CAE), Distributed Database Connection Services (DDCS), the IBM Data Propagator family and IBM DataJoiner.

Client support is provided for the following environments:

- OS/2
- DOS
- Windows (3.1, 95 and NT)
- AIX
- HP-UX
- SUN Solaris
- Sinix
- Santa Cruz Operations (SCO)
- OS/400
- MVS/ESA
- VM/ESA
- VSE/ESA

For all environments except DOS, Macintosh and Windows 3.1, there is also the option of installing complete database management systems (servers).

The actual relational database management functions are provided by the DB2 servers and by IBM DataJoiner.

Appendix B. Bibliography

- *DataPropagator Relational Guide*, SC26-3399.
- *DataJoiner for AIX: An Introduction*, GC26-8243. To obtain a copy of this document from the World Wide Web, see URL: http://www.software.ibm.com/data/datajoiner/
- *DataJoiner Implementation and Usage Guide*, SG24-2566.
- *Net.Data Programming Guide*. This document is only available on the World Wide Web at URL: http://www.software.ibm.com/data/net.data/docs

Where to Get Additional Information

For additional information about data replication, data management, and the DataJoiner product, visit URL: http://www.software.ibm.com/data/dbtools/dbsmwp.html/

For additional information about DB2, visit URL: http://www.software.ibm.com/data/db2/

Appendix C. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AnyNet
AS/400
CICS
DataHub
DataJoiner
DataPropagator
DB2
Distributed Relational Database Architecture
DRDA
IBM
IBMLink
IIN
IMS
MVS
MVS/ESA
Open Blueprint
OS/2
OS/400
RACF
VM/ESA
VSE/ESA

The following terms are trademarks of other companies:

DCE	The Open Software Foundation
Encina	Transarc Corporation
Excel	Microsoft Corporation
FoxPro	Fox Holdings, Incorporated
HP	Hewlett-Packard Company
Java	Sun Microsystems, Incorporated
Lotus Notes	Lotus Development Corporation
Macintosh	Apple Computer, Incorporated
NT	Microsoft Corporation
OSF	Open Software Foundation
SCO	The Santa Cruz Operation, Incorporated
Solaris	Sun Microsystems, Incorporated
Sybase	Sybase Corporation
Windows NT	Microsoft Corporation
X/Open	X/Open Company Limited

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Appendix D. Communicating Your Comments to IBM

If you especially like or dislike anything about this paper, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply. Feel free to comment on specific error or omissions, accuracy, organization, subject matter, or completeness of this paper.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send comments by FAX, use this number:

  United States and Canada: 1-800-227-5088.

- If you prefer to send comments electronically, use one of these ID's:
  - Internet: **USIB2HPD@VNET.IBM.COM**
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**

Make sure to include the following in your note:

- Title of this paper
- Page number or topic to which your comment applies

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC23-3925-01