DataBlade Extensions for INFORMIX-Universal Server

Michael A. Olson
Informix Software
mao@informix.com

Abstract

In September of 1996, Informix Software released the first version of its new object-relational database management system to developers and partners. This system, called INFORMIX-Universal Server, supported a new way of building and deploying database applications. Developers could write software modules, called DataBlade extensions, that extended the database server with knowledge of new types and operations. This paper describes the architecture of INFORMIX-Universal Server and how it supports DataBlade extensions. The paper describes the way that DataBlade developers and application developers interact with the server. Finally, it describes a set of DataBlade extensions available for the server at the end of 1996.

1. Introduction

In September of 1996, Informix Software released the first commercially-available version of INFORMIX-Universal Server (IUS) to partners and developers. IUS is an *object-relational database management system* (ORDBMS). It is an enhancement of Informix's relational product. It allows software developers to write code that extends the database system to manage new data types, and to provide new operations on existing data types. This code can run in any of a number of address spaces, including in the address space of the database server itself. These extensions can be written by Informix, its customers, or third parties, and operate in the same way, regardless of origin.

The software modules that enhance IUS are called *DataBlade extensions*. To date, about thirty DataBlade extensions to IUS have been completed by Informix and third parties, and many more are in some stage of development. Informix believes that the database server market will change dramatically as a result of the appearance of DataBlade extensions, and has invested heavily in the technology to support them.

From Informix's point of view, DataBlade extensions have several advantages over the middle-ware and client-side systems that are common today. First, the DataBlade modules are snap-in components, and are easy to write and use. Second, the components interoperate easily, because the database server and the database user interact with them all in precisely the same way. Third, allowing extensions to run directly in the database server, if desired, can make applications much faster. No data copying is required.

This paper describes, in some detail, how DataBlade extensions are designed and built, and how IUS finds them and uses them while executing user queries. The paper begins with a fairly high-level discussion of the architecture of IUS, and how the architecture supports DataBlade deployment. The next section covers the various ways that DataBlade code interacts with the database server, and what that means for DataBlade developers. After that, the paper examines IUS and DataBlades from the point of view of application developers. Finally, it briefly describes the variety of DataBlade extensions supported by IUS at the end of December, 1996.

2. IUS Architecture

INFORMIX-Universal Server is an enhancement of Informix's core relational product, the On-Line Data Server. Users who choose to do so can ignore all available DataBlade products and run IUS as a full-function, high-performance relational engine. However, the real value of IUS is its ability to manage new kinds of data in new ways, and DataBlade code is the key.

IUS contains a number of components that allow it to find and use DataBlade code on demand. This section describes the architecture of INFORMIX-Universal Server at a high level, and introduces the important components of a database management system.

2.1. Conventional RDBMS components

The major vendors of relational database management systems all organize their systems in the same way. Naturally, differences in implementation can dramatically affect the speed or feature set of any particular vendor's product, but all such systems rely on the same services to do their jobs.

The major components of a relational database management system are a query parser, a planner and optimizer, an execution engine, and a collection of access method code. There are a number of other important system components, but these four are the important ones for the current discussion.

2.1.1 The query parser

When a query arrives at any relational server, the query string is delivered to the query parser. The parser converts the string into an internal representation of the query. This internal representation converts constants, type names, table and column names, operators, and function names that appear in the query string into values that can be more conveniently manipulated in the computer's memory. Typically, a parser produces a tree data structure that captures the request of the user. The parse tree ignores differences in white space, commenting, and so forth, that can make two identical queries look different.

2.1.2 The planner and optimzer

After parsing, relational systems pass the parse tree to a module that does query planning and optimization. This code is responsible for picking the best possible strategy for satisfying a user query. For example, a query that searches a corporate database for all employees who make twenty thousand dollars a year may be satisfied in a number of different ways. The system could scan the entire employee table, comparing each salary to the desired value. Alternatively, if there is an index on the salary column of the employee table, the index can be consulted, and will produce exactly those records that qualify. Which of these strategies is best depends on the comparison in use, the amount of data that is likely to be returned, and the performance characteristics of the system running the query. The planner and optimizer consider the variables and choose the best query execution plan.

2.1.3 The query executor

Once the optimizer and planner are finished, the query execution plan is handed to an execution engine for evaluation. That engine iterates over all the data according to plan, and produces answers for the user. It is the query executor that performs restrictions, projections, and join operations.

2.1.4 The access methods

The query execution engine needs access to data stored in tables and indices. Most systems use an abstraction called an *access method* to do this. Simply speaking, an access method is the code that understands the layout of data on disk, and knows how to fetch and update the data while keeping the layout consistent. For example, a tree-based indexed access method like a B-tree stores records in groups on pages, and must keep inter-page pointers consistent when updating the index.

2.2. Support for extensibility

Because INFORMIX-Universal Server is built on the On-Line Data Server relational code base, it has all the modules described above. However, IUS makes several substantial changes to some of those modules to make DataBlade code useful.

The biggest difference between IUS and conventional relational engines is that IUS does not hard-code knowledge of a fixed set of data types into the engine. For performance reasons, most relational vendors have built special-purpose code for handling integers, floating point numbers, and character strings into their engines. As a result, these systems cannot operate on new data types, because the system does not have support for them built in.

IUS includes built-in knowledge of the popular relational types, to guarantee good performance. However, when confronted with a data type or function name it does not recognize, the system will consult database tables to find code that supports the type or the requested function. All known types and functions are described in database tables. Adding a new type or function to the system simply requires writing some software and registering its presence in the tables.

The next several subsections describe the changes to various relational modules in IUS to support DataBlade code.

2.2.1 Parser changes

The main change to the query parser is that it looks up type and function names in tables whenever necessary. When confronted with a name that had not been built into the system, the On-Line Data Server would raise an error. IUS, in contrast, searches the system tables

to determine whether or not the type or function is supported by DataBlade code.

As a result, a legal parse tree can refer not just to the types and operations built into the server when it was shipped, but to any type or operation that has been defined since.

2.2.2 Planner and optimizer changes

There are a number of important changes to the planner and the optimizer. In general, running user-supplied code during the execution of a query can dramatically affect the space and time the query requires. The optimizer and the planner must consider these variables when they choose a query plan.

DataBlade developers can specify the cost of the functions that they write at development time. During query planning, IUS will consider these costs, along with its knowledge of data distributions, number of records likely to be touched, and so forth, to produce a good execution strategy.

2.2.3 Function resolution and execution

During query execution, IUS may need to run a number of functions over data stored in tables. Some of these functions may be built into IUS, and some may be supported by DataBlade code. From the user's point of view, these two cases are indistinguishable; the SQL looks the same, regardless of who wrote the supporting code. IUS must decide which case applies, and must be able to find, load, and execute DataBlade routines on demand, exactly as if they were built into the engine.

Function resolution and execution in IUS is handled by a new module, called the *function manager*. The function manager consults the table of functions that are known to the system, and decides whether the desired routine is built into the engine or is provided by a DataBlade extension.

If the function is built into the engine, it is called directly, and the answer is returned to the query executor.

If the function is defined by a DataBlade extension, then the function manager decides how to call the routine. DataBlade code can be written in a number of different languages, and the calling strategy depends on the implementation language and the calling conventions imposed by that language.

To call DataBlade code written in C, the function manager uses the operating system's support for dynamic linking to move the DataBlade code into the address space of the server process. Another operating system-dependent call returns the address of the function of interest. Given that address, the function manager can call the routine directly.

To call DataBlade code written in Java, the server reads the code into memory and invokes the Java Virtual Machine (which itself operates as a DataBlade extension) on the desired function.

The IUS function manager is designed to take advantage of extensibility itself. It is easy to add support for new languages, like C++, or remote method invocation services, like CORBA, by writing a new kind of DataBlade extension called a language manager.

2.2.4 Access method interfaces

One final important difference between IUS and relational products is the way in which the INFORMIX-Universal Server interacts with access method code. As described earlier, an access method is the code that manages data storage in tables or indices. Access method code can control the layout, storage, and retrieval of records. B-tree indices and ordinary tables are two examples of access methods familiar to relational system users.

Relational systems hard-code support for a small number of access methods, and only support storage and retrieval of a fixed set of data types. It is impossible for ordinary developers to add a new kind of index to relational systems, because those systems have no idea how to open, search, and update the index. In addition, relational systems cannot store new data types in existing indices and tables, because the access method code itself hard-codes support for a small number of data types.

IUS diverges from relational systems in two ways.

First, there is an abstract *access method interface* that allows the system to use any access method. This means that developers can define new ways of storing tables or managing indices, and IUS can take advantage of them for storage and retrieval of data. A number of current and future DataBlade developers are writing access methods for IUS.

Second, access methods are themselves type-extensible. IUS introduces an abstract way of storing and operating on values in an access method. The access method code asks the system for the supporting routines for a particular data type, and then uses those routines to operate on values of that type. This means that a DataBlade developer can register routines that allow values of a new type to be searched for using B-tree indices, or any other indexed access method that the developer wants to make available.

3. DataBlade Development

The previous several sections have described DataBlades from the point of view of the INFORMIX-Universal Server. A much more interesting view is that of actual or potential DataBlade developers: How must developers write their code to support execution in the Informix server?

There are two ways to consider the interaction between the server and DataBlade code. The first is the way in which the server will call DataBlade functions, and the interfaces that the developer must provide. The second is the way that the DataBlade code can call server routines, and the interfaces that the developer may use.

3.1. The DataBlade Developers' Kit

When developers define new data types for IUS, they must declare a suite of support functions required by the server to handle byte swapping on heterogeneous networks, conversion to and from textual representation, and several other operations. In general, these routines are fairly simple, but tedious to write. Their formal parameters are the same for all new types.

In addition, when a developer creates new routines that are callable from SQL, the supporting C or Java code must conform to the calling conventions imposed by IUS. Again, the conventions are simple, and the prototypes are predictable.

In order to speed up development of DataBlade code, Informix provides a tool called the DataBlade Developers' Kit. This kit is a collection of GUI tools for defining types, functions, access methods, and other database objects, generating code, and packaging DataBlades for release and installation.

The DataBlade Developers' Kit automatically generates the supporting routines, including routine bodies, for new data types. As a result, a new type developer need not define the basic support routines required by IUS. The developer can concentrate on writing the functions that operate on values of the new type.

In addition, the Kit automatically generates files that contain function prototypes for the new routines defined by DataBlade extensions. Developers need only fill in the routine bodies.

The effect of the Kit is to isolate the developer from having to learn the calling conventions and interfaces in use by IUS for calling DataBlade code.

3.2. The DataBlade API

The second important suite of interfaces for DataBlade developers is the set of routines that are callable inside the engine to perform important operations.

IUS includes its own memory manager, file manager, and other services. There is a public API for developers who need these services. This API is called the DataBlade API. Developers who want to run DataBlade code in INFORMIX-Universal Server should use these routines, rather than operating system calls.

The API is broken into groups of routines that handle resource allocation (including memory and file management), query execution and result binding, error detection and reporting, and data conversion support for transmission of values between different CPU architectures, among others. Whenever the DataBlade API offers functionality that is also provided by operating systems, the API mimics the arguments and return types defined by POSIX. The intent is to provide developers with a familiar set of interfaces that operate well inside the database server.

Developers need only pay attention to the API when they need to interact with the database server or do resource allocation. Generally speaking, no changes are required to code that does computation. DataBlade routines look just like routines that appear in application or library code. No special programming skills are required to write DataBlade functions or to use the DataBlade API.

4. Application Development

Programmers that develop client applications against the INFORMIX-Universal Server can use the same tools and techniques that they do when they write applications against strictly relational database servers, including Informix's On-Line Data Server.

When a DataBlade extension is registered for use at a customer site, the registration code runs a collection of SQL scripts that update the database catalog with its new types, functions, and access methods. That process extends the SQL-3 query language with the type, function, and access method names that the DataBlade code provides. As a result, the interface to the DataBlade extension is the same as the interface to the rest of the database services. Users write queries in SQL, and IUS executes them.

All of Informix's programming tools have been modified to handle extended SQL-3. In particular, developers who write C++ code, use SQL embedded in a programming language, or other application development tools can continue to do so with IUS.

Applications may mix and match data types and functions defined by multiple DataBlade extensions. The database server correctly identifies and invokes the required support code on demand. The application developer need not be aware of which features are built into the server, or which are provided by which DataBlade extension.

When an application executes a query that returns values of a new data type, the server will deliver the results in either textual or binary format, just as it does for conventional types like integers and floating point numbers. The values are delivered in the byte order of the client. DataBlade developers, application developers, or other third parties can also supply client-side components like ActiveX controls or Java applets to handle display and manipulation of values on the client.

5. DataBlade Extensions for IUS

As of December 1996 (which was the publication deadline for this paper), twenty-nine DataBlade extensions were shipping for INFORMIX-Universal Server. Almost all of these products were developed by third parties, using the DataBlade API against early developer releases of IUS. The functionality and feature sets of the products vary widely, but all are interesting products that provide sophisticated services not available in conventional relational databases.

The next several sections break that initial suite of DataBlade extensions into functionally similar groups, and describe some of the features provided by each group. In practice, there is no clear demarcation among groups. Text search, for example, is useful on the Web. However, for convenience in discussing features, the categorizations make some sense.

The extensions are currently in use by a variety of end users and systems integrators building applications on top of IUS.

5.1. Text and document management

Several DataBlade extensions provide text searching and document management support. These products do high-speed searching for documents, returning those that match query criteria in ranked order, as well as conventional document management, including revision control and tracking, access control, and so on.

The text offerings are generally used as horizontal technology. They apply to no problem domain in particular. Rather, most customers who purchase INFORMIX-Universal Server have documents they want to store, search, and retrieve, and want to integrate that service with the rest of their data processing strategy. As a result, the text DataBlade extensions are extremely popular.

5.2. Digital media

There is a large number of DataBlade extensions that manage popular digital media types like video, audio, and still images. These DataBlade extensions handle not just storage and retrieval, but also content-based search of the media types. For example, a user can submit a sound clip as an example, and scan the database for similar sound clips as part of an SQL query. For the time-dependent types like audio and video, DataBlade vendors have built interfaces to streaming storage servers, as well as providing management of values stored directly inside the database system.

These DataBlade extensions are typically used in two different ways. First, every Web-based application includes multimedia data, and so most Web applications built on top of IUS include a collection of digital media DataBlade extensions. Second, there are a number of specialized markets for digital media management for which IUS products are being developed. An example is high-end video production, where editors would like an intelligent storage system that permitted content-based retrieval of videos and access to high-performance video storage systems.

5.3. Web, Internet, and Intranet

In addition to storing and searching documents, videos, audio clips, and still images, Web-based applications need special services from the database system. Several DataBlade extensions have appeared to address that need.

The extensions that are focused on the Web generally handle electronic commerce, including secure, reliable transmission of offers and bids, encryption of data to be transmitted across insecure networks, authentication, access control, and tracking.

5.4. Financial services

There is a small, but important, group of DataBlade extensions that provide special services for the financial markets. In general, the efficient storage and retrieval of stock ticker information, aggregation and interpretation of those values, and interfaces to real-time data feeds are important for these markets.

5.5. Data warehousing

Several DataBlade vendors have built extensions for sale to customers with large data warehouses. The DataBlade vendors and their customers build data mining applications on top of IUS running these DataBlade extensions. Services provided include duplicate detection and reporting in very large databases of name and address information, aggregation and interpretation of large volumes of data, and statistical analyses of raw or summary data.

5.6. Spatial data handling

Another important market for developers of IUS DataBlade extensions is the geographical information services sector. These developers build tools that store, search, retrieve, and interpret spatial information. For example, most companies store address information for their employees and customers, and most would like to search their databases to find customers who live near new stores, or employees who live near each other for ride sharing programs.

A large number of companies have developed DataBlade extensions for spatial data. These extensions typically do address geocoding (turning an address into a latitude and longitude for location on a map), distance, containment, or overlap searches, and access methods that support fast spatial searching.

Like the vendors of text DataBlade extensions, the spatial vendors believe that they provide horizontal technology. No single market is a focus for these vendors. Rather, many applications, including data warehouses, Web-based catalogues, financial markets, and others want to augment their databases with spatial capabilities.

6. Conclusion

INFORMIX-Universal Server extends the architecture of relational systems in a number of conceptually simple but important ways. As a result, IUS is able to load and execute DataBlade extensions on demand to search new kinds of data for new properties of interest to users.

DataBlade extensions can be developed by anyone, and can run in the server, in the client, or in a middleware application server. This provides developers and customers with the flexibility to tune their system for performance and manageability when they develop new applications.

Developers of DataBlade extensions can use Informix-supplied tools, like the DataBlade Developers' Kit, to write software easily that can run in the database server. IUS defines a simple suite of interfaces, called the DataBlade API, for use by DataBlade developers. As a result, DataBlade development is no more complicated than conventional application development.

Informix's customers can purchase IUS and any collection of DataBlade extensions, and can install and administer them as a unified system. Application developers interact with DataBlade extensions in the same way that they interact with the rest of Informix's data management services. They write SQL queries using a variety of development tools, and have a wide range of choices for client-side manipulation and display of new data types.

The ease with which DataBlade extensions can be developed, combined with the large new markets that they allow software vendors to address, have produced an enormous number of DataBlade extensions for IUS in very short order. As of December 1996, twenty-nine were shipping, with many more in development.

Informix believes that the DataBlade model represents the future of database management, and has invested heavily in architecture and business support to encourage DataBlade development. Informix is planning for thousands of new DataBlade extensions to appear over the next few years.