Implementing the Spirit of SQL-99

Paul Brown
INFORMIX Software
20th Floor, 300 Lakeside Drive.
OAKLAND, CA , 94612. USA .
01-510-628-3765
brown@informix.com

ABSTRACT

This paper describes the current INFORMIX IDS/UD release (9.2 or Centaur) and compares and contrasts its functionality with the features of the SQL-99 language standard. The IDS/UD product supports most of the innovative features of the new standard, although at this time INFORMIX's query language implements a slightly variant syntax. In addition, we review the experience of early technology adopters working with object-relational DBMSs (ORDBMS). We argue that their difficulties indicate that the SQL-99 language standard as it currently stands is going to be largely irrelevant, in that developers will not use it when implementing next-generation information systems, and DBMS vendors and their partners will need to go beyond it in their products.

Keywords

INFORMIX, object-relational database, SQL, language standards.

1. INTRODUCTION

Release 9.2 of the INFORMIX Dynamic Server with Universal Data option (IDS/UD) implements most of the data model features standardized in the SQL-99 language. In this presentation we provide an overview of this functionality. We include brief detours describing how two of these features -- the extensible language manager and multi-representational types – are implemented.

However, an interesting lesson of our first two years shipping an object-relational DBMS has been that SQL-99 style development presents a series of quite different challenges from what was encountered with SQL-92. We summarize these difficulties, and argue that what they indicate is that the SQL-99 language standard will be largely irrelevant to developers of next-generation systems.

2. INFORMIX IDS/UD

The following table presents a partial list of the interesting SQL-99 features supported by IDS/UD. INFORMIX was among the first vendors to provide extensibility and object-oriented data model as features its DBMS. At the time we began developing this functionality we focused on implementing the then draft SQL-3 standard. Since then the syntax of the standard has changed substantially.

This leaves us in the position of supporting almost all of the new features of SQL-99 through slightly non-standard syntax. For obvious market reasons, we anticipate rapidly moving our ORDBMS query language closer to the standard now that it is a more fixed target.

User-defined Types (UDTs)

- SQL-92 Built-in Types and Expressions
- ROW Type
- COLLECTION Type
- DISTINCT Type

User-defined Routines (UDRs)

- EXTERNAL Routines in 'C', Java (SQLJ Part 1) and C++[1]
- Internal routines in INFORMIX Stored Procedure Language(SPL)
- Mutator, Observer, Operator, Constructor expressions
- UDR Overloading

Inheritance and Polymorphism

- ROW TYPE Inheritance
- Table Inheritance
- Polymorphic Queries

Query Language Features

- COLLECTION Derived Tables
- Closed Query Expressions

Figure 1. Partial List of SQL-99 Features in INFORMIX 9.2

Many of the non-core aspects of the standard – like the SQL/MM spatial data types and function, and the temporal extensions – can be implemented using these features. For example, the SQL-99 `Period` data type that represents a fixed interval in the time-line is handled as a UDT. In order to support these features efficiently IDS/UD provides interfaces that let extension developers overload our data management services like sorting, indexing, replication and so on.

[1] On Microsoft platforms only.

2.1 Beyond the Standard

Release 9.2 of IDS/UD includes functionality that goes beyond the standard in several areas. Therefore, INFORMIX is working to affect the direction of the standard as it evolves. The following list summarizes the additional, INFORMIX specific functionality.

- Open Storage Manager Interfaces
- Extension Language Integration
- OPAQUE Types
- User-defined Aggregates

Figure 2. Extra-Standard Features of the 9.2 Release

3. Implementation Examples

In this section we present more detailed explanations of two features: how IDS/UD handles UDRs implemented in multiple language, and how we support data types of extremely variable length.

3.1 Language Manager Extensibility

Developers using IDS/UD can implement extensions using a variety of procedural languages: INFORMIX's proprietary stored procedure language, a semi-compiled language like Java, or 'C' compiled into shared library binaries. What is common to all of these extensibility alternatives is that the user-defined code runs within the same memory address space as the DBMS process. This design achieves optimal performance because it minimizes the overhead incurred when the ORDBMS invokes the user-defined code.

Early in our design process we decided that the engine needed an abstracted interface that would support the addition of multiple language environments. This generalized extension mechanism consists of a set of procedure calls – which must be implemented in 'C' – to handle argument marshaling, procedure invocation, return values and exceptions. Developers integrating fully 'sand-boxed' environments like Java or Visual Basic must also map system calls – requests for resources like memory, I/O and thread management – to their IDS/UD equivalents.

The mechanism is general enough that it allows us to link the `JAVA.LIB` library shipping with various Java distributions into the DBMS address space. Our initial implementation of Java in the database took a more orthodox approach by running the Java virtual machine in its own address space and communicated with it through shared memory. Tighter integration yielded significant performance benefits. We are repeating the process for COM interfaces on Windows™ systems.

1.2 Multi-representational Types

Managing variable length data types presents some difficult problems. For example, one of the data types we manage in the server is the SQL-99/MM `st_polygon`. In our implementation, polygons representing US state and territory boundaries vary in length by several orders of magnitude. Boundaries for square states like Colorado and New Mexico are 72 bytes long, while boundaries for states like Texas and Maine can take up hundreds of kilobytes. In the table below we present a histogram of the number of points per boundary (at 1.4 degree minimum edge).

Range of Points	Number of States
Less than 10	25
10 to 20	26
20 to 40	7
40 to 80	6
80 to 160	3
More than 160	1

Figure 3. Histogram of Points in Geographic Boundaries

Efficient processing of queries involving spatial data requires a two-phase approach. In phase one, we use an approximation – usually a bounding rectangle – for a rough check to exclude obviously false matches. Then we perform an exhaustive check on the approximate matches in phase two. Storing data for polygons like Texas in the table's row multiplies scan times for the entire data set. But storing all polygon data separately also implies significant overhead, as the DBMS must visit the large object storage to retrieve several smaller objects.

The solution is what we call a multi-representational type, which is made possible through the OPAQUE type mechanism. Developers implementing an extended type can use interfaces provided by the server to specify a threshold value, and when the object exceeds this limit the object's data is moved to large-object storage. In the `st_polygon` example we always store the bound-box and some meta-data in the record, and optionally page the polygon data into large-object storage, depending on its size.

4. THE RELEVANCE OF SQL-99

A language standard is relevant to the extent that it is both *useful*, and *used*. By useful, we mean the extent to which the standard addresses the problems it is intended to solve; portability of applications and of skill-sets. By used, we mean – loosely – the extent to which it is adopted and followed by vendors and application developers. In the remainder of this paper, we argue that, based on the experience of our early adopters, the SQL-99 standard is not useful when building applications that include interesting UDTs, and will not be widely used.

The intention of what follows is not to diminish the achievement of SQL-99. The standard is a thorough, rigorous document: the product of an enormous and well-intentioned effort. Also, it is quite possible that ORDBMSs can be used as slightly more general RDBMS systems; i.e. the advantages of SQL-99 are the modularity and re-use that can be attributed to features like ROW TYPES and inheritance and the extended types defined in SQL-99/MM.

If SQL-99 has a failure it is that textual query languages and procedural APIs – which were conceived in the days of character terminals -- are no longer the most appropriate model to use when addressing complex object management in an era dominated by graphical user interfaces. Instead, developers benefit from adopting a more component-centric view of the overall information system, and using the ORDBMS as a framework for these components.

In the following sections we describe several problems we have encountered building information systems using extensible DBMSs.

4.1 The Problem of Multiple Extensions

In SQL-92 databases the schema consists of a collection of inter-dependent tables, each of which consists of a set of columns. The SQL-92 language standardizes a simple type system for these columns and a set of expressions for the query language. All that the SQL-92 developer needs to know is their schema design and a few hundred pages of a SQL textbook[2][6]. But in an ORDBMS, the database includes a great many types and functions. For example, the GIS extensions provided by INFORMIX's partners include about fifty data types, and perhaps one thousand functions. A SQL-92 style developer using an extensible DBMS is obliged to remember the correct spelling of each of these function's identifiers, their argument order, and when several functions are combined into a single query expression they also need to know the function's return type.

As a result, working with SQL-99 is very hard. Developers using the INFORMIX ORDBMS commonly request that we provide some kind of schema browser that presents the database's schema objects – tables, columns, types and functions – in a GUI. For example, consider developing a system that mixes geographic types, digitized microscope image data and pattern recognition functions in a single database. Queries against such a system might be crafted by hand, but a far more efficient alternative is for the ORDBMS vendor to provide a tool to do so instead. This diminishes the utility of the standard for application developers and vendors. Developers no longer need to rely on their knowledge of what the standard says to write queries, and vendors no longer need to adhere to it in order to be useful to developers.

A variety of commercial and research systems have demonstrated how graphical techniques can usurp much of the functionality of a query language.[1][7] In these systems, and in tools like INFORMIX-Visionary®, the interface generates queries and displays their results without the user being aware what SQL expressions are involved.

4.2 The Problem of the API

The current state of the art application programming interfaces (APIs) are either embedded language approaches – ESQL/C, embedded Java (SQLJ Part 0) – or call level interface APIs – SQL-99/CLI[5], ODBC, JDBC. Either style is really only useful when the type system of the DBMS has a close correspondence to the type system of the host language program. With SQL-92 this is almost always true. Historically, the SQL language was intended for embedding within COBOL or 'C', and more recently 4GLs. The small number of exceptions – SQL's `DECIMAL` type has no obvious equivalent in 'C', for example -- are handled by the vendor's client libraries.

But with SQL-99, the host language program may not know the return types from a query until the ORDBMS executes it, and the external programming language will almost certainly not know what to do with the kinds of data that are returned by the ORDBMS. For example, consider the following query:

```
SELECT Histogram ( E.Salary ),
       E.Department
  FROM Employees E
 GROUP BY E.Department;
```

Figure 4. SQL-99 Style Query

In this example, the `Histogram` aggregate returns a complex data type. The first problem is that the definition of this data type may change between invocations of the query. In other words, every ORDBMS query is a dynamic query. This is not an issue with SQL-92 systems where every expression has standardized properties. The second problem is that the language standard does not (and should not) specify low-level data structures (although, in cases like SQL-99/MM[4] spatial types binary standards do exist). The standard does provide the means to map or convert a server-side object into a client-side object. But for tools vendors who will need to provide access to databases containing user-defined types with arbitrary data structures such mapping does not solve the problem.

All of the standard APIs are *data-centric*. That is, they are designed to manage data values. But this is not enough with an extensible DBMS. There needs to be a mechanism for the ORDBMS to pass entire *interfaces* back to the host language: that is, the means to manipulate query result objects on the client side without the external program knowing a-priori what the return results will be.[2] A more useful ORDBMS API standard would be *component* or *object-centric*, not just data-centric.

In order to implement systems using ORDBMSs, developers and vendors will be obliged to go beyond the SQL-99 standard, which gives no guidance in this regard.

4.3 The Problem of Porting SQL-99 Queries

A primary objective of a language standard is to provide portability of applications and skill-sets between products. With the same data set on two RDBMSs, the same queries will return the same results. But our experience has been that even porting applications between different extension libraries with our *own* ORDBMS poses significant challenges.

The problem is more semantics than syntax. Queries that return one result with one set of text extensions return different results on another, even though the syntax is identical. For example, given a library of medical articles, when one vendor's extension functions are asked to return all articles containing the concept 'hypertension' it might return 100 unordered articles in a few seconds. The same query expression, using another vendor's extensions, may return 200 articles ordered by the degree of conceptual relevance in a minute or two. Neither answer is wrong.

[2] JDBC 2.0 provides a means of mapping a query's return result to a Java class, and that class might be loaded into the VM independently of the database interface.

What this indicates is that the SQL-99 standard cannot achieve the goal of application portability between standards compliant systems. Language standards fix syntax. Even developers who manage to meet their application requirements while sticking religiously to the letter of the standard will find their system exhibits different functionality in different DBMSs.

4.4 The Problem of Architectural Diversity

ORDBMS products are more architecturally varied than RDBMS products. Further, some of them include support for data manipulation algorithms that are neither part of the standard relational model, nor mentioned in the standard (sampling, on-line aggregation, result-set size limits etc). Also, some ORDBMS vendors have a strategic focus on business partnerships to supply extensions to the core DBMS. It is in the interests of these third-party creators of extensions that their products be differentiated too. Both of these factors will motivate a divergence of database functionality.

This indicates that ORDBMSs are likely to become increasingly proprietary, and that developers will need to use these proprietary extensions to achieve their development goals. The set of common functionality, defined within the standard, will be a tiny sub-set of what each ORDBMS provides. And developers who restrict themselves to using it will not be the ones building new systems.

5. DBMS INTERFACE DIRECTIONS

Over time, DBMS products have evolved considerably beyond their original purpose, which was to store data and respond to external queries. Modern RDBMSs include sophisticated active DBMS facilities and can host procedural logic implementing complex business processes. ORDBMS technology continues this trend. It turns the DBMS into a *component framework*. The fact that an ORDBMS includes scaleable, transactional data storage functionality can be seen as entirely incidental. For example, we are beginning to see our customers use the ORDBMSs as a middle-ware server. In these circumstances, the query language is simply a high-level notation for reasoning about the components the system manages.

The alternative to a textual query language is a more abstracted interface that presents developers and even end-users with conceptual-level information system objects. They then combine and manipulate these objects in the user-interface. All of the logical principles that have guided DBMS interface design -- the emphasis on a dynamic programming and declarative expressions – and on supporting technologies – query processing and optimization – remain relevant. The important point is that the syntactical notation used to express these queries is entirely hidden. It might well be based on SQL-99, but it need not be. A precedent for this kind of usage pattern can be seen in CASE tools that generate DDL for different RDBMSs, and report-writer tools that generate DML. What matters to the developer or end-user is the functionality of the interface. Less important is the syntax of the SQL it generates.

6. CONCLUSION

In this paper we have briefly described the extent of INFORMIX's support for the SQL-99 language standard. The IDS/UD product supports most of the innovative features of SQL-99: user-defined types and functions, object features like inheritance and polymorphism, and new query language features like closure. Although INFORMIX's syntax is a slight variation on the standard's syntax, we anticipate adapting to the standard quite quickly.

The second part of this paper discussed several observations about how developers are using SQL-99 style features, and concluded that the language standard will not be useful to such developers.

7. ACKNOWLEDGMENTS

Thanks to Mike Ubell and Charles Campbell, INFORMIX's representatives on the SQL standard committee, for their comments and clarifications on the standard.

8. REFERENCES

[1] Bloesch, Anthony. C. and Halpin, Terry A. "ConQuer: A Conceptual Query Language." 15th Int. Conf. on Conceptual Modelling. Berlin, Germany . 1996.

[2] Date, C. J. and Darwen, Hugh. A Guide to The SQL Standard Third Edition. Addison-Wesley Publishing Company. Menlo Park, CA. 1994.

[3] ISO Draft International Standard Database Language SQL – Part 2: Foundation December, 1998.

[4] ISO Working Draft SQL Multimedia and Application Packages: Part 2: Full-Text September 1995.

[5] ISO Working Draft Database Language SQL – Part 3: Call-Level Interface July, 1998.

[6] Melton, Jim and Simon, Alan R. Understanding the New SQL: A Complete Guide Morgan Kaufmann Publishers. San Francisco, CA. 1993.

[7] Microsoft Access Version 97. Microsoft Corporation. Redmond, Washington. 1997.