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. INFORMIX and Illustra have been shipping DBMSs implementing the spirit of the SQL-99 standard for five years. In this paper, we review our experience working with ORDBMS technology, and argue that while SQL-99 is a huge improvement over SQL-92, substantial further work is necessary to make object-relational DBMSs truly useful. Specifically, we describe several interesting pieces of functionality unique to IDS/UD, and several dilemmas our customers have encountered that the standard does not address.

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 and query language features standardized in the SQL-99 language specification. In this presentation we provide an overview of this functionality.

However, the more interesting lesson of our first five 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. These difficulties indicate that the SQL-99 standard can and should be improved upon. We include brief descriptions of why, and how, several useful non-standard features of our engine are implemented, and describe several areas where further standardization effort is required.

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 an object-relational data model as features of its DBMS. This list reflects the functionality of the currently shipping ORDBMS product.

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 functions, or the T-SQL extensions can be implemented using these features. For example, the `Period` data type that represents a fixed interval in the time-line can be 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. In addition INFORMIX has been very aggressive in adopting API standards as they emerge: JDBC, SGML and OLE-DB.

3. BEYOND THE STANDARD

INFORMIX's 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. In this section, we describe several IDS/UD specific technologies, explaining briefly why they are necessary and describing how they are implemented. We also describe several problems encountered by developers using ORDBMS technology. Taken collectively, these

[1] On Microsoft platforms only.

issues indicate that in order to remain inter-galactic database speak, SQL-99 requires substantial further work.

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 viewed 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. What these aspects of the standard do not address however, is the diverse data ecology that must be addressed by modern application developers.

3.1 Open Storage Manager

The IDS/UD product supports an open storage manager, called the Virtual Table Interface or VTI. This provides the means whereby external or remote data sources may be integrated within the ORDBMS and presented to developers as tables. Such an interface is considerably more sophisticated than an interface like OLE-DB; it needs (optional) interfaces to logging and locking facilities, cost estimators for query planning and more sophisticated metadata handling. For example, VTI allows us to create interfaces to XML documents, ERP middleware and even distributed components with transient data (a JINI toaster, for example). This means that we can use SQL to place orders in the ERP system to buy toasters from e-commerce vendors as replacements for currently broken ones.

```
INSERT INTO ERP_Orders
( Vendor, Product, Quantity )
SELECT W.Vendor_Id,
       T.Product_Part_Id,
       1
  FROM Web_XML_Suppliers W, JINI_Toasters T
 WHERE T.IsBusted
   AND T.Product_Part_Id IN
                          W.Products_Supplied;
```

Figure 2. Query Involving Three Eternal Data Sources

Standards for supporting this kind of multi-database integration are sorely needed, and we strongly endorse the work of the SQL/MED (Management of External Data) sub-committee in this area.

3.2 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. Vendors failing to adopt this architectural model will be at a significant performance disadvantage [9].

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 executable's address space. We are repeating the process for COM interfaces on Windows™ systems.

SQL-99 does not include a standard way to integrate new language environments. It would be a tremendous boon to language vendors like Borland, Franz Inc., and Lucent to integrate Delphi, LISP and Inferno into DBMSs in a standard way.

3.3 OPAQUE 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

The solution requires what we call a multi-representational type, which is made possible through the INFORMIX OPAQUE type mechanism. Developers implementing an OPAQUE type in IDS/UD 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.

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. In the general case, variable length objects like `st_polygon`, or digital signal data, and many business objects, are best handled using the (non-standard) OPAQUE type representation.

Therefore, in our opinion, an OPAQUE type or encapsulated component interface is a highly desirable feature missing from the current standard.

3.4 User-defined Aggregates

Among our most successful ORDBMS customers are several using the ORDBMS as an analytical engine. The kinds of analysis they perform are poorly supported by SQL-92 systems. For example, they want to perform spatial aggregates like *ConvexHull* or problem domain specific analysis like *ValueAtRisk*. In the general case such customers need user-defined aggregates: extension mechanisms that permit them to efficiently (i.e. with parallelism) scan a large number of data objects and compute some result. The requirements for such an interface are quite well understood [6] although at this time only INFORMIX has implemented one.

For SQL to be successful in decision support applications, a user-defined aggregate standard is necessary.

3.5 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][7]. 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 support much of the functionality of a query language.[1][8] 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.

SQL would benefit from a standard graphical schema browser and query builder, and standard exchange format for graphical queries.

3.6 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, in which a `Histogram` user-defined aggregate returns a complex data type.

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

Figure 3. SQL-99 Style Query

The problem with embedding this query into any host language is that the data type returned by the aggregate may change between invocations. For example, this aggregate may be part of a package of statistical or analytic extensions that must be upgraded occasionally. To achieve the same physical and logical abstraction that was possible in SQL-92, every ORDBMS query needs to be treated as a dynamic query. Dynamic queries are less of an issue with SQL-92 systems because every expression has standardized properties, and every data type has its host language equivalent.

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. The JDBC 2.0 standard provides the means to map server-side objects into pre-existing client-side objects. But for tools vendors who will need to provide access to databases containing user-defined types with arbitrary data structures – and arbitrary visualization algorithms -- such mapping does not solve the problem.

To be useful, the SQL-99 APIs should evolve into *component* or *object-centric* interfaces.

3.7 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. Both queries are – to the extent that they can be – compliant with the standard.

What this indicates is that 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. What seems to be needed are SQL/MM style efforts for a multitude of other domains: financial data, audio, video, digital signal data, currency, etc.

4. 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 middleware server. In these circumstances, the query language is simply a high-level notation for reasoning about the components the system manages.

The problems cataloged in Section 3 are all low-level development issues. In our opinion, the correct way to address all of them is to step back and rethink the nature of the DBMS interface. Perhaps the major problem with SQL-99 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.

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 a graphical user-interface. The interface is component-centric, and when it receives objects from the DBMS, these objects include interfaces letting the client system render the object on the screen.

All of the logical principles that have guided DBMS interface design; i.e. the emphasis on dynamic programming and declarative expressions and supporting technologies like 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.

5. 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. We concluded that the language standard, while a great leap forward, still needs considerable work before it can be as widely used as the database community would like it to be.

6. ACKNOWLEDGMENTS

Thanks to Mike Ubell and Charles Campbell, INFORMIX's representatives on the SQL standard committee, for their comments and clarifications on the standard, Mike Stonebraker for his advice on an earlier draft, and Caroline Cleaves.

7. 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] Jaedicke, Michael. and Mitschang, Bernhard. "On Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS" Proc. of SIGMOD 97. Seattle, USA. 1997.

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

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

[9] Stonebraker M, and Brown, P. Object-Relational DBMS: Tracking the Next Great Wave. Morgan Kaufmann. San Meteo, CA. 1998.