Object Query Standards

by
Andrew E. Wade, Ph.D.
Objectivity, Inc.
drew@objy.com
www.objectivity.com

ABSTRACT

As object technology is adopted by software systems for analysis and design, language, GUI, and frameworks, the database community also is working to support objects, and to develop standards for that support. A key benefit of object technology is the ability for different objects and object tools to interoperate, so it's critical that such DBMS object standards interoperate with those of the rest of the object world. Starting with a discussion of the new issues objects bring to query standards, we present the efforts of various groups relevant to this, including ODMG, OMG, ANSI X3H2 (SQL3), and recent merger efforts feeding into SQL3.

I. What's different with Objects?

The introduction of objects into database standards raises several new issues. In fact, it raises the need for new kinds of interfaces. In addition to the familiar declarative query language, the object approach presents the possibility to integrate the interface with an object language, so the objects used in the language are handled automatically by the database. This allows programmers to work entirely in the object language and avoid translating between database structures and language structures. In addition, the declarative language for defining object types can be much broader, allowing many different kinds of structures. Finally, the query language, too, can allow for accessing such different structures. So, there are potentially interfaces for type definition, for query, and for integration with object programming languages, all with broader varieties of types.

More than this difference in structure, the fundamental nature of objects presents new capabilities to be addressed in all three of these types of interfaces. These include:

- encapsulation (operations)
- identity
- references
- collections

A basic difference between an object and traditional data is that the object also includes operations. The user of the object may invoke these operations and may access externally visible state information (attributes), both of which may be implemented using internal state information which is invisible externally. The type definition for the external user, then, must include a specification of the attributes and of the operations. To implement objects, there must be a way to specify the implementation of the operations as well as any necessary internal state information.

Another basic difference is identity. Traditional databases store only data. It may be queried and modified associatively (by value), and users may create keys based on those values to locate desired records, but the only meaning and the only access is by these values. With objects, the system maintains a concept of identity. Even if all the state information (all the attribute values) of the object changes, the user may still uniquely access objects by internally generated and maintained object identifiers (OIDs), and the user may ask the system if any two objects are in fact the same (identity operation). The object interfaces to databases, then, must provide some means of accessing these identities.

Above, we mentioned that the user of an object may access the object's externally visible attributes and operations, but there is also another externally accessible characteristic, the object's relationships. For binary relationships, a simple one-to-one relationship can be viewed as

an object-valued attribute within each of the two related objects. Many-to-many relationships can be viewed as attributes on each side with many object values, or with a collection of object values. The type definition interface must support a means to declare such relationships, and the query and language interfaces must support means to create, remove, and traverse them. N-ary relationships might also be of interest.

Collections can be thought of as multi-valued attributes, which individual values, or members, of the collections comprising various, possibly complex, data types, as well as objects. Various sub-types of collections are possible, commonly including a bag or multi-set (no ordering, no uniqueness requirement), a set (a bag in which elements are unique), a list (a bag with unique ordering), an array (a bag with direct access by element ordinal), all possibly varying in size dynamically. Again, the type definition language must provide a way to define such collections, and the query and object language interface must provide a way to create, delete, read, and update such collections.

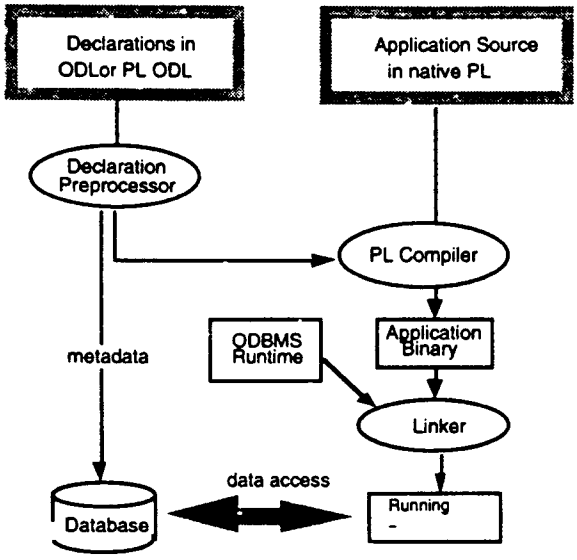
2. ODMG

The Object Database Management Group (ODMG), a consortium of object database vendors, has created all three of the above described interfaces in order to jointly propose them to OMG (Object Management Group), ANSI (American National Standards Institute), ISO (International Standards Organization), and other relevant user organizations. In particular, the book ODMG-93 contains four interfaces:

- Object Definition Language (ODL)
- Object Query Language (OQL)
- Binding to C++
- Binding to Smalltalk

For the interface to an object language, ODMG chose to use pre-existing languages, C++ and Smalltalk, and bind them to the database object model, so that programmers in those two language could define objects normally, and the database would automatically support them.

For object definition, ODMG began with OMG's Interface Definition Language (IDL), and added the extra capabilities needed for object databases, including declaration of collections, many-to-many bi-directional relationships, keys, and (optional) extents. Combined with IDL's ability to define attributes and operations, this allows defining any objects. Also, since all the additions to IDL are defined as semantically equivalent to the generated methods (e.g., traversal methods for relationships), all ODL objects immediately produce IDL interfaces, so are accessible in the world of OMG's standards, including the Common Object Request Broker Architecture (CORBA, a method dispatcher), and the Object Services and Facilities.



Using an ODBMS with ODMG

For query, ODMG began with the SQL-92 (sometimes know by the nickname SQL2) basic query capability, the SELECT statement, and added the ability to apply queries to any object or collection of objects, and to invoke methods within the queries. This is defined by a simple, closed, typed functional language. The type of the result of the query may be a scalar (including tuples), an object, or a collection of objects, with type combination rules specifying which operations on which types produce which other types. Invocation of operations is syntactically the same as attribute references, with the optional addition of parameters. Traversal of relationships is accomplished via a dot (a.b) syntax. Also, syntax is added to specify collections, including lists, sets, bags, and arrays.

```
• Select x
from x in faculty
where x.salary
>x.dept.chair.salary

• sort s in
(select struct (name:
x.name, s:x.ssn)
```

```
from x in faculty
where for all y in
x.advisees:y.age<25)
      by s.name

• Chair.salary

• Students except TAs

• list (1,2) + list (count
(jse.advisees), 1+2)

• exists x in faculty
[1:n]: x.spouse.age<25
```

Except for these additions (traversal dot, operation parameters, and collection notation), OQL looks very much like SQL2's read-only query (SELECT statement). In the most recent version, V1.2, ODMG attempted to make OQL completely compatible with SQL2 in the sense that any SQL2 query would be a legal OQL query, with the same syntax, semantics, and result. Although this was achieved to a large extent (perhaps 90%, some suggest), it is not compatible in all cases. There are cases in which an OQL query and the syntactically identical SQL2 query will produce different types of results. SQL2 always works in terms of table types, but OQL can produce collections. So, for example, the SQL3 query

select f(x) from R x where p(x)

returns a table of type

multiset(row(type(f(x))

while in OQL it returns

multiset(type(f(x))

SQL2 contains a long and involved list of special cases, while OQL has a short, simple definition. So far, ODMG has stopped short of adding all these special cases and casting to tables.

3. OMG

The Object Management Group (OMG) is a software consortium producing interface specifications to allow interoperability among objects and object tools. They began with CORBA, a mechanism for a user to invoke operations on remote objects. Each object registers with the CORBA dispatcher the operations it supports by means of IDL (see above). Then, OMG added definitions for interfaces to several services, each in terms of IDL, including:

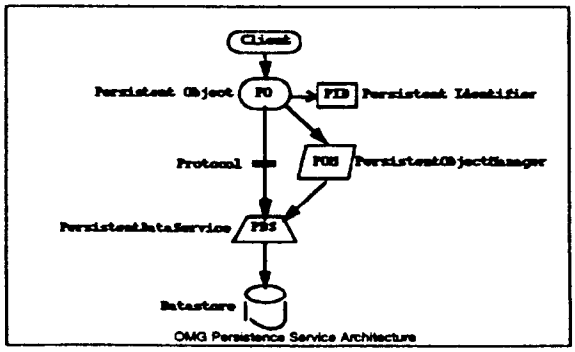
- Naming
- Persistence
- Relationships
- Transactions
- Query

etc.

OMG is now working with ISO Open Distributed Processing (ODP) to input their CORBA interfaces to the formal standardization process.

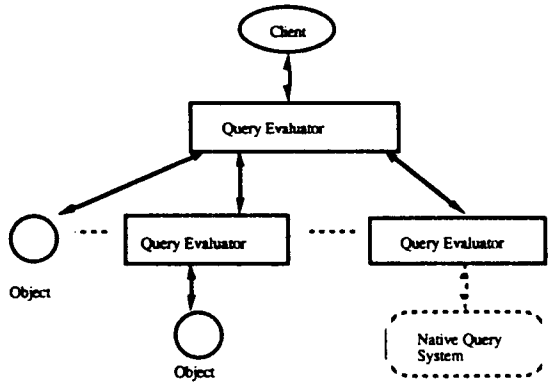
Several of the OMG services overlap the domain of DBMSs (Database Management Systems). In fact, a DBMS can be viewed as supplying these several OMG services all in one package, in such a way that the package also includes recovery, common buffering and indexing for performance, etc. Another way to look at the relationship of OMG to DBMSs is via a layering. The OMG Transaction Service, e.g., is compatible with the X/Open X/A protocol for two-phase commit, as is commonly used by transaction manager products to coordinate atomic transactions across multiple DBMSs and other resource managers.

The persistence service provides an interface that allows a client to request that an object be made persistent, including storing the object's current state, and restoring a previously stored state. This is rather different than the direct object language interfaces or query interfaces discussed above and used in ODBMSs, but can still be implemented in terms of DBMSs. In particular, the persistence service includes a Persistent Object Manager (POM) that provides the interface to the client and to the objects being stored, Persistent Identity Service (PID) that generates identifiers to be used by the client to later restore state, and Persistent Data Stores (PDSs) where state is retained. The backend protocol between the POM and PDS may be any of three different ones defined in the specification, two of which are designed to flatten object state into linear streams that can be stored in a file (sequential byte stream) or record-oriented database, while the third calls out the ODMG-93 interface to directly store the objects, as objects, in a DBMS that supports such an object interface.



OMG Persistence Service Architecture

The OMG Query Service specification defines a nested federation of query servers. The user sends a query to the query service, which may then pass on sub-queries to other query servers. In that way, if part of the query applies to objects stored in a DBMS, that DBMS may directly execute that sub-query, using whatever internal optimization mechanisms it may have. The top-level query service assembles the results of all such sub-queries, together with direct evaluations of query predicates on stand-alone OMG objects, and returns this result, as an OMG scalar, object, or collection of objects, to the user.



OMG Query Service Architecture

The language used to express the query may be any language that the user and server agree on. Languages are labeled using OMG's object typing mechanism. The specification requires, though, that any compliant implementation support at least one of the following:

- a specified subset of SQL2 (approximately SELECT plus INSERT, UPDATE, and DELETE)
- OQL
- a subset of OQL, restricted to simple queries and sets, but including operations.

The first is intended for the widest compatibility with commercial query-related tools. The second provides complete object functionality, and is close to a superset of the first (see previous section for exceptions). As we see in the next section, work continues to make this an exact superset. The third query language is provided for use by other OMG services that want a very simple predicate evaluation language, including the ability for predicates to invoke operations, but implementable with a small footprint. By defining this as a subset, such other services can avoid defining different predicate languages, and users who wish to advance from such a simple service to full query service have a natural upgrade path.

OMG continues to work on other areas relevant to DBMSs. In particular, they are currently evaluating submissions for the Collection Service, which will extend the primitive collection interface included in the Query Service, and may be useful for users of DBMSs.

4. SQL3

ANSI X3H2, in cooperation with ISO/IEC JTC1/SC21/WG3 DBL, is working on the next version of the SQL standard, sometimes called by the nickname SQL3. The current draft includes many new capabilities, including two very significant additions: object functionality, and a computationally complete language (PSM). This draft is still changing, but substantial results have been achieved.

The addition of PSM provides a full language, with control flow, procedural operations, and function resolution. This is a different approach than OMG and ODMG, who chose, instead of creating their own languages, to map to other languages (C++ and Smalltalk for ODMG, those plus C and Ada for OMG, so far). However, there is no conflict. Both OMG and ODMG expect to add other language bindings in the future. So, once PSM settles into its final form, it will likely be straight-forward, and almost certainly possible, for OMG and ODMG to produce mappings or bindings into SQL3 PSM, just as they have with other languages in the past, and will likely do with yet more other languages in the future.

The addition of object capabilities includes the ability to define and access Abstract Data Types (ADTs), which have much the same functionality as OMG or ODMG objects, including externally visible attributes and operations. SQL3's own language (PSM) is used to implement the internals of these objects, both state and operations. The intent is to add "oo-ness" to rows in tables, as opposed to OMG's and ODMG's general object approach. These rows may contain ADTs. A mechanism to reference ADTs in other rows exists, providing a means to uniquely identify such a row (rather than searching for it by its contained values). This is the basis for a concept of identity, similar to OMG's PIDs and ODMG's Object References (Ref<>).

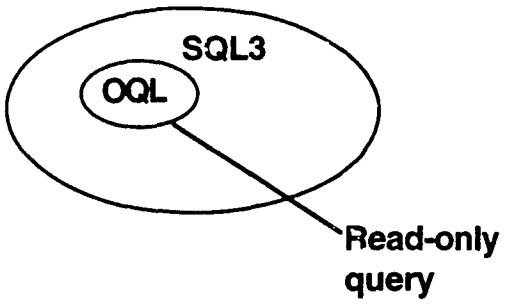
Some of the issues still being debated by the SQL3 committees include update and authorization. While ADT operations are the natural object approach to changing the state of an object, SQL2 requires that update of data be performed via an explicit request, such as UPDATE. One approach would be to use "copy" semantics, so that the user retrieves an object (or perhaps many objects) into some private space, operates on them using ADT methods, and then updates the database by invoking an explicit UPDATE statement. This has the advantage of similarity to the SQL2 approach, and a natural place to check user authorization at the time of UPDATE. It is, however, unnatural to an object user and seems to require the user to re-specify all changes in the UPDATE statement. This latter might not even be possible in some situations. After such updates, an explicit COMMIT is still required to make the changes durable and visible to other users.

Another approach is to use "reference" semantics, in which ADT operations are viewed as directly changing objects in the database. No extra UPDATE statement is required. This approach is more natural to object users and easier for the user, avoiding the repeating of all operations, but requires some other solution to checking user authorization. Although SQL3 already includes an EXECUTION permission check, providing authorization at the level of object or table would require some extra mechanism, such as guard functions that are invoked at the time of operation invocation.

Queries in SQL3 include the exact query ability of SQL2 (not surprisingly), along with additional ability to invoke methods and traverse relationships. Again, this is much like ODMG's query (which is used also by OMG). Where ODMG queries may apply to and result in objects and collections, SQL3 queries are limited to tables. There are syntax differences, also; e.g., where ODMG uses the dot to indicate traversal, SQL3 has used double-dot (a..b) to avoid conflict the with the use of dot in SQL2 for naming (catalogue.table).

5. Efforts to merge

Although many different DBMS products and architectures are likely to be developed, and this is desirable for users, it is not desirable that there be two (or many) different query languages. Rather, it would be better if at least some common core declarative query language could be agreed, still allowing the diversity of implementation underneath.



This is exactly the goal of a joint working group formed between X3H2 and ODMG. A small group of key technical contributors from both organizations have been working together, and are making progress. The goal, as indicated in the figure, is not to force either model or either standard to become the same as the other, but rather to seek a common read-only query language. SQL3 will certainly contain more, including PSM, and ODMG will certainly contain more, including ODL and language bindings.

To date, significant progress has already been achieved. Several changes were made to ODMG's OQL to accommodate this merger. These, now published in V1.2, include:

- Addition of null data type
- Addition of table data type
- Several detailed syntax adjustments

Although the goal of V1.2 was to achieve 100% compatibility with SQL2's query, this was not quite achieved (some have described it as 90%). The difference is due mainly to the typed approach of OQL (see above). Where SQL queries only tables, and returns only tuples, OQL queries any objects and collections, and returns tuples, objects, or collections. There are also many special cases developed over the years of standardization process within SQL.

The merger group next looked at SQL3, analyzed the object model differences, and concluded that only two needed to be addressed to achieve the common query language goal. The first was to tie the OID to the object, rather than leaving it tied to the table location, which would result in objects changing OIDs as they move. This has been accomplished by eliminating the separate concept of object, and including references to ADTs in rows (LHR77).

To address the latter difference, and add to SQL3 the ability to query objects and collections, the merger group has agreed on and begun to execute the following approach:

- Identify BNF elements in OQL that result in scalars, objects, and collections
- Within the BNF for SQL3, wherever scalars, objects, and collections appear, insert the corresponding OQL expression
- Identify and resolve any syntax conflicts
- Identify and remove syntax redundancies

The relevant OQL syntax comprises a couple pages of BNF. The relevant portions of the SQL3 BNF, after removing unnecessary intermediate terminals, is also only a few pages. Thus, this task looks achievable. Of course, it will be necessary to change syntax at some points, but hopefully, with the model issues resolved, the syntax issues should not be overwhelming.

OMG's Query Service anticipated this merger, and is structured to accept the results immediately, via the tags that allow passing new query languages. Further, it is expected, when the SQL3 specification is finalized, that OMG will adjust to accommodate it directly.

References

1. SQL documents

The SQL-92 standard is
ISO/ANSI SQL
Database Language SQL
International Standard ISO/IEC 9075:1992
American National Standard X3.135-1992
American National Standards Institute
November 1992.

The SQL3 Working Draft documents that are publicly available can be obtained from any National Standards Body actively involved in ISO/IEC JTC1 standardization. See ftp://speckle.ncsl.nist.gov/isowg3/dbl/doclog.txt for a list of online documents.

The following documents are the March 1995 Working Drafts for extensions to Database Language SQL, ISO/IEC 9075:1992:

SC21 N9443 Description of Status of SQL3 Parts

SC21 N9462 SQL3 Part 1: Framework
N9463 SQL3 Part 2: Foundation
N9464 SQL3 Part 3: Call Level Interface
N9465 SQL3 Part 4: Persistent Stored Modules
N9466 SQL3 Part 5: Language Bindings
N9467 SQL3 Part 6: XA Specialization

Parts 2, 3, and 4 are furthest along - Part 5 is equivalent to material in SQL-92. All of the Object SQL facilities are in Part 2.

2. ODMG

R. Cattell, Ed, The Object Database Standard: ODMG-93, ISBN 1-55860-302-6,
Morgan Kaufmann Publishers, San Francisco, California, 1993. (800) 745-7323
In Europe, available from Afterhurst, +44-273-748427, +44-273-722180 fax
orders@mkp.com, http://www.mkp.com, http://www.omg.org

3. OMG

http://www.omg.org for introductory information
ftp://ftp.omg.org/pub/docs/doclist for list of documents
Also published by OMG,
Object Management Group
Framingham Corporate Center
492 Old Connecticut Path
Framingham, MA 01701
(508) 820-4300
(508) 820-4303 fax

Persistent Object Service,
ftp://ftp.omg.org/pub/docs/doclist/94-10-7.ps
Query Service,
ftp://ftp.omg.org/pub/docs/95-1-1.ps