Object-Oriented Extensions in SQL3: A Status Report

Krishna G. Kulkarni
Tandem Computers Inc.
10100 N. Tantau Ave.
Cupertino, CA 95014
email: kulkarni_krishna@tandem.com

SQL3 is a new database language standard being developed by both ANSI X3H2 and ISO DBL committees for the last three years. SQL3 is upward compatible with SQL-92, the current ANSI/ISO database language standard, and is targeted for completion in 1997. SQL3 extends SQL-92 in many significant ways, one of the major extensions being the addition of an extensible, object-oriented type system. This talk will describe the current status of SQL3 type system.

Apart from the ability to create flat tables, SQL-92 offers no facilities to define complex data types. In contrast, SQL3 offers a fairly rich type system based on the notion of *abstract data types* (ADTs) that let users capture the application-specific behavior as part of the database. ADT definitions correspond to a set of attribute and routine (procedure/function) definitions. An ADT is completely encapsulated in that only its behavior is visible outside the type definition, but not the implementation of its attributes and routines. Visibility of attributes and routines is further controlled by different levels of encapsulation associated with the attribute or routine, which can be one of PUBLIC, PRIVATE, or PROTECTED. ADT routines can either be implemented using SQL3 procedural extensions or using code written in external languages. User-defined unary or binary operators can be defined to operate on instances of ADTs.

ADTs can be either *object ADTs* whose instances are associated with unique, system-assigned identifiers, or *value ADTs* whose instances have no such unique identifiers associated with them. Further, ADTs can be related in subtype-supertype relationships, where an ADT can be a subtype of multiple supertypes. Instances of subtypes can be substituted wherever instances of supertypes are expected. Resolution of overloaded routines is based on a form of non-selfish algorithm where types of all arguments in a routine invocation are considered in making the decision. Dynamic binding is also supported whenever compile-time binding is not possible. However, type checking is always performed at compile time.

ADT definitions can be parameterized, wherein a family of types can be defined with a single type definition. Collection types such as, SET, MULTISET, and LIST types are provided as built-in parameterized types. Operations allowed on tables are also applicable on instances of collection types. The notion of *distinct types* provides the capability to create a new type based on an existing type.

ADTs (including built-in collection types) can be used as data types of variables, parameters of routines, attributes of ADTs, or columns of tables. Persistent instances of ADTs can be stored in columns of tables and can be queried using the familiar SQL constructs. Queries can refer to attributes and functions on ADT instances and a form of path notation provides for logical navigation.

Plans for future work include adding array and record (tuple) types, refining the notion of type extents and making the specification as consistent and complete as possible.