CQL++: A SQL for the Ode Object-Oriented DBMS

S. Dar
N. H. Gehani
H. V. Jagadish
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

CQL++ is a declarative front end to the Ode object-oriented database. It combines a SQL-like syntax with the C++ class model. CQL++ provides facilities for defining classes, and for creating, querying, displaying, and updating objects. Classes and objects created using CQL++ can freely be intermixed with those created using O++, the primary programming language of Ode. CQL++ gives its users a relatively straightforward interface, by hiding from them such O++ details as object-ids, public and private members of objects, and the implementations (bodies) of member functions.

CQL++ is based on an object algebra that preserves the closure property of SQL. CQL++ queries operate upon sets of objects and return sets of objects. CQL++ supports a rich collection of set types to maximize functionality and expressive power. CQL++ allows the user to define new classes, which can include computed attributes and make use of multiple inheritance. The user can supply default and null values for attributes, and indicate how class objects are to be displayed.

1. INTRODUCTION

Ode is a database system and environment based on the object paradigm. It offers an integrated data model for both database and general purpose manipulation [2, 3]. The database is defined, queried, and manipulated in the database programming language O++ which is based on C++ [26]. O++ borrows and extends the object definition facility of C++, called the class. O++ provides facilities for creating and manipulating persistent objects, defining sets, and iterating over sets and clusters of persistent objects. It also provides facilities for specifying constraints and triggers.

CQL++ is an adaptation of SQL [19] to the Ode object-oriented model. We were motivated to design a SQL-like interface to Ode because SQL is by far the most popular database interface. Typical database users are not likely to use a programming language such as O++. A declarative interface to Ode based on SQL would appeal to such users.

Several designs have been proposed for adapting a relational language such as SQL to nested and recursive data models [20, 22, 24]. A major strength of these proposals is that they are based on well defined data models. These models, like the relational model itself, comprise a formal description of the data structures of the database and the algebraic operations on these structures. In contrast, object-oriented languages [5-7, 12, 14, 16, 18, 23] are usually defined "operationally" rather than algebraically or axiomatically. CQL++ strives to combine the advantages of the structural and operational models. It is based on an object algebra that preserves the closure property of SQL, and applies uniformly to sets of objects, regardless of their type. Orthogonally, type specific operations are used to construct new objects and to display existing objects. These operations are specified by the user, as part of the class definition.
CQL++ provides facilities for creating, querying, displaying, and updating objects. CQL++ queries operate upon sets of objects and return sets of objects. Various types of sets are supported: clusters and cluster hierarchies are collections of persistent objects, while temporary clusters are used to store objects for the duration of a session. In addition to views, CQL++ introduces object and set variables as handles for manipulation of objects or sets of objects. CQL++ allows users to define new classes and supports the definition of computed attributes and the use of multiple inheritance. The user can supply default and null values for class attributes, specify constraints, and indicate how class objects are to be displayed.

An important goal in the design of CQL++ is to ensure its integration with O++. Although, CQL++ does not support the full definitional capabilities of O++, it can be used to formulate queries against a database defined and populated using O++. Similarly, O++ users can manipulate objects of classes defined using CQL++. CQL++ tries to minimize the details of the Ode object model users must know, while maximizing the functionality of O++ available to them. It hides from users such programming language details as object-ids, public and private members of objects, and the implementation (bodies) of member functions. See [8] for more detail.

2. BACKGROUND

The object model of Ode is based on C++ [26]. The database schema is a set of classes. A class is similar to an abstract data type: it has public members (components), which represent the interface to the class, and private members, which represent the implementation of the class. The members of the class can be data items or functions (methods). Instances of a class are called objects. Two special categories of member functions are constructors, used to create and initialize objects, and destructors, used to delete class objects. As an example, we give below the definition of class person in O++. (We have omitted some definitions, such as those for types Name and Address, and the bodies of member functions).

O++ extends the C++ object model by allowing objects to be persistent. Every persistent object has an object-id that uniquely identifies it. We also refer to this object-id as a pointer to a persistent object. Each class has an associated cluster in which its persistent instances reside. Clusters can be partitioned into subclusters.

```cpp
class Person {
    int ssn;    // social security number
    Date birth_date;

    public:
        Name name;
        char sex;
        Address addr;
        persistent Person *kids[MAX_CHILDREN];    // a set

    Person(Name n, char s, int snum);    // a constructor
    int ssnum() const { return ssn; }
    int age() const
        { return (today().year - birth_date.year); }    // a computed attribute
    virtual void print()
        { cout << ssnum() << ", " << name << ", " << sex; }
    virtual void identify() { cout << name; }
};
```