Querying Object-Oriented Databases SQL-style*

Michael Kifer†
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
kifer@cs.sunysb.edu

Yehoshua Sagiv‡
Department of Computer Science
The Hebrew University
Jerusalem 91904, Israel
sagiv@cs.huji.ac.il

Won Kim
UniSQL, Inc.
9390 Research Blvd.
Austin, TX 78759, U.S.A.
kim@unisql.com

March 31, 1995

Abstract

We present a novel database query language, called *XSQL*, that provides uniform access to object-oriented *and* relational data. The language has many features not found in earlier proposals; it is easier to use and has greater expressive power.

XSQL is built around the idea of extended path expressions that substantially generalize those proposed in [33, 39]. These path expressions can not only do joins, selections and unnesting, but they can also be used to access meta-information about the database. As a result, database schema can be explored naturally, without relying on user's knowledge of system tables that store schema information. The semantics of XSQL is rigorously defined; it is an adaptation of the first-order formalization of object-oriented languages from [24, 23, 25].

Views in XSQL can be specified and manipulated in a much more uniform way than in other proposals. In fact, it seems to be the only language where, by analogy with relational languages, non-trivial object-based views can be defined as queries. The notions of a type and type-correctness are given precise meaning. This accommodates a wide variety of queries that might be deemed well-typed or ill-typed under different circumstances. In particular, we show that there is more than one way of settling the issue of type correctness.

*A preliminary report on this work has appeared in [21].

†Work supported in part by the NSF grants CCR-9102159 and IRI-9404629.

‡Work supported by NSF grant IRI-8722886, AFOSR grant 88-0212, and a grant of IBM Corporation.

1 Introduction

In recent years, several papers [3, 4, 16, 18] have proposed query languages for object-oriented databases. However, these languages fail to capture (and often do not attempt to deal with) many aspects of the object-oriented model. In this paper, we present a new query language that incorporates features not found in earlier languages. The proposed language, henceforth referred to as $XSQL$, is easier to use and has greater expressive power than previous languages. XSQL's data model uniformly supports relational and object-oriented data and the query language provides uniform access to both these types of data. It is not our goal here to introduce the full-fledged syntax of XSQL. Rather, we use the familiar SQL-style notation to illustrate certain philosophy in designing object-oriented languages, a philosophy some of whose elements earlier appeared in [14, 24, 26, 25, 23]. A subset of XSQL was commercially implemented by UniSQL, Inc., as part of its database product.

Before going into the discussion of the the features of our language, we should point out some of the differences between the object-oriented model and the relational model. The different features of these two models induce different modes of representing information and querying it. A detailed discussion of these issues is found in [26]; we will describe some of the important aspects through an example.

Suppose that a database includes information about engines and their types (e.g., turbo engines, diesel engines, etc.). In a relational database, there would likely be an attribute, *EngineType*, having the various engine types as its possible values. In contrast, in an object-oriented database, there would be a class, *Engines*, having the various engine types as its subclasses. This is a fundamental difference, because it shifts the information about engine types from the data to the schema. For example, suppose we want to know all the engine types registered in our database.[1] In the relational model, we simply project onto the attribute *EngineType.* In the object-oriented model, we have to interrogate the schema rather than the data, and there is hardly any language for doing that. For another example, suppose that we want to find all vehicles having a diesel engine. In the relational model this is done by applying a simple selection to the attribute *EngineType.* In some object-oriented query languages the user is forced to express it as a rather cumbersome query that looks more like a join than a selection.

The above example shows the need for features not available in relational query languages. In particular, since an object-oriented schema is likely to have much more information than a relational schema, querying the schema (as well as querying the data without a complete knowledge of the schema) becomes an important issue. We also need to deal easily with nested structures.

$XSQL$ provides these and other features through *path expressions.* Although the idea of path expressions is not new—it first appeared in [33] and had many incarnations since then—our *extended* path expressions have the following features and expressive power not found in earlier incarnations of this idea.

1. Path expressions may have variables that range over classes and attributes (and even methods) rather than data, and so it is possible to query data without having complete knowledge of the schema. Earlier query languages for object-oriented databases completely lack any similar feature. The languages in [12, 11, 13, 29] have some related capabilities, but they were designed for the relational model. Note that, in spite of having variables that range over classes, attributes,

[1] Actually, this example is rather intricate. One may want to know all the engine types that are currently installed in some vehicles, or one may want to know all the engine types that exist, including those that are currently not installed in any vehicle. The language we propose can handle easily each one of these possibilities.