ProSQL: A Prototyping Tool for SQL Temporal Language Extensions

James Green, Roger Johnson

School of Computer Science, Birkbeck, University of London,
Malet Street, London WC1E 7HX, UK.
JamesGreen@liberata.com , r.johnson@dcs.bbk.ac.uk

Abstract. This paper describes ProSQL, a novel prototyping tool to support the development of extensions to SQL. ProSQL provides a simple way to prototype the features of a proposed extension and thus provide a proof of concept. Further, it provides proposers and reviewers of extensions with a clearer view of their positive and negative features. The approach adopted has been to build a wrapper around an existing database management system, in this case Microsoft Access, and to provide a collection of interfaces with which a designer can define extensions to the basic relational database.

1. Introduction

This paper describes ProSQL, a novel prototyping tool to support the development of extensions to SQL. The work has been carried out in the context of temporal extensions to SQL but the authors believe that the approach adopted can be readily applied to a wide range of other potential extensions to SQL.

Many temporal extensions to SQL have been proposed although few have been implemented. In studying temporal extensions it became clear to the authors of this paper that claims were being made about their ease of use and productivity which had often not been substantiated by controlled experiments.

The approach adopted has been to build a wrapper around an existing database management system, in this case Microsoft Access, and to provide a collection of interfaces with which a designer can define extensions to the basic relational database. The facilities include new data types, new comparison operators as well as temporal features. The authors recognize that there are limitations to the range of language extensions that can be implemented in this way. However, their experience suggests that the range is sufficiently large to make this approach useful for language developers and HCI researchers.

2. Motivation

The starting point for this research was the authors' previous work in extending SQL to handle spatio-temporal data by means of intervals. Assumptions have been made in

the literature about the ease of use of temporal extensions [16]. Conceptual models have been presented which are claimed to be more intuitive to the user [17, 5].

The authors of this paper have a long term concern with assessing the usability of SQL extensions. Though a formal definition of usability does not exist there is a general agreement on its constituent parts [20]. These are best described as efficiency, effectiveness and satisfaction. Efficiency can be determined by speed of learning and accuracy; effectiveness by memorability and rate of errors whereas satisfaction is a subjective measure.

Designers of temporal extensions have only built implementations that support their own extension and have used them primarily to demonstrate feasibility [14] and to evaluate their extension in isolation [5]. In addition, gaining access to many of these extensions by independent researchers is not possible and, as the interfaces inevitably differ anyway, comparative analysis of extensions using their own implementation is not possible as the difference in the interface would invalidate the result [2,21].

This has provided the motivation for developing ProSQL, a prototyping tool for SQL extensions that can support a basic simulation of a range of different SQL extensions. The application allows the user to define extensions and their associated operations thus providing a tool that can be used in usability tests and other experiments at an early stage in the design without incurring the time penalties and associated development costs of a full implementation. It also eliminates the problems associated with different interface behaviour and shows that a basic simulation of a range of SQL extensions can be achieved relatively easily.

3. Previous Work

So far as the authors are aware Pro SQL is the first attempt to build a prototyping tool to support the development of extensions to SQL in this way. ProSQL offers a simple standard interface to a range of potential SQL extensions which support comparative studies between competing alternatives.

Since the implementation of a common interface for the languages being compared is beyond the scope of most usability researchers, the great majority of experiments have used paper and pen tests. These have been shown to be effective [10,11, 19]. However, Yen and Scammel found the use of an interface could yield different results to pen and paper although their experiment compared a text based language, SQL, with a graphical one, QBE [21].

Tests using query language interfaces have also been used in previous experiments [21, 3]. Prototype applications have been used in usability tests of database query languages [6]. However, these experiments were undertaken when the languages were already commercially available and, therefore, too late to influence SQL design materially.

The purpose of most SQL extension implementations appears limited to demonstrating feasibility. While they allow users to experience the language no reports of systematic testing appear in the literature.

4. Application

The definition of an SQL extension involves a sequence of relatively straightforward steps. First the extension has to be named. Once this is done the new data types, a test database, predicates and their mappings to the underlying SQL are defined followed by the specification of an extension's functions and mapping them to those built in to the underlying database. (If a required base function does not exist it has to be programmed before it can be mapped to an extension's function.). At any stage in the extensions definition the researcher can define and populate databases and use them to test their extension definition.

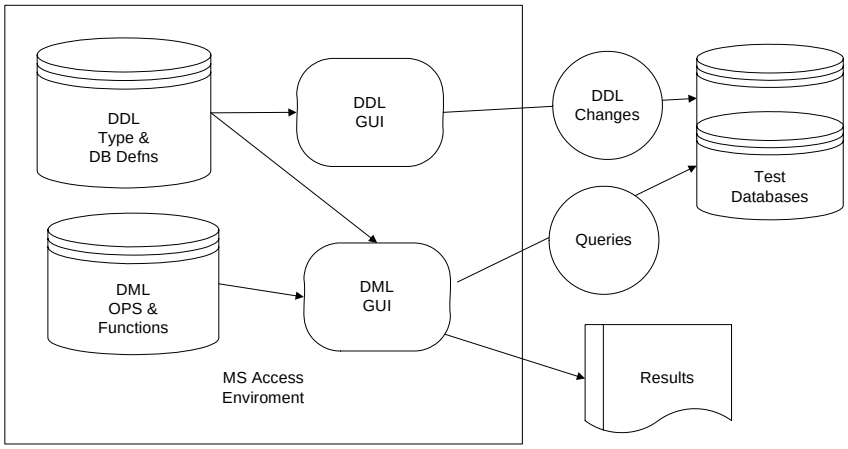

Fig. 1. ProSQL Architecture

The prototyping tool has been developed using an approach similar to that of other implementations of query language extensions using a two tiered system. In this approach, the base layer is a conventional RDBMS [18, 15]. The actual RDBMS used by ProSQL is Microsoft ACCESS. The benefit of using this is that it is cheap, widely used and does not need any specialist DBA skills to maintain it. The emphasis is on the production of a tool that can be easily installed and used without requiring additional specialist knowledge to maintain it or extra software to use it. This approach suffers from some of the problems described in [15] as the simulation of an extension using a wrapper has a detrimental affect on performance.

The outer layer is a wrapper that forms the interface with the user and maps the extension's view of a query to one that can be processed by the underlying RDBMS. It provides support for a data definition language (DDL), a data manipulation language (DML) and uses a set of system catalogues to support non-atomic data types, SQL extensions and operations.

The SQL language can be divided into three sub languages which are the data definition language (DDL), data manipulation language (DML) and data control language (DCL). Each of these will now be described in turn.

4.1 Support for a DDL

The DDL is a GUI application used to define databases, normal, valid time period and event tables and support for tuple time stamping, attribute time stamping (ATS) and explicitly defined period attributes. When a database and its relation schemes are defined it is not bound to a specific extension recorded in the RDBMS. The user selects the extension that is used when they want to query a database. The failure to bind a database to a given extension is not the result of an oversight. Query languages that provide support for the same relational model, data types and tables can be used on the same database structure in an experimental situation.

4.2 Definition of New Non Atomic Data Type

Before an extension's language can be modeled its base types must be available to the application. This includes its non-atomic data type. All the sub types of a generic domain like an interval, for example DATE INTERVAL, INTEGER INTERVAL, have to be declared separately, which renders their definition simple but relatively tedious. New data types are declared using a collection of atomic types supported by the underlying RDBMS, along with a character string for each element of the type that can be appended to an attributes name. This is used to identify part of the data type in the base RDBMS storage system.

The structure used above to map a statement belonging to an extension to an equivalent structure supported by the underlying RDBMS is typical of the approach used throughout the application as it usually allows a set of simple procedures to be used to construct the mapped statements.

4.3 Database definition

The next step in the definition of an extension is the definition of a database whose relations utilise the extension's novel data types and relation's properties and it provides a test database when the extension is being defined.

Relations are defined using the GUI displayed in *Figure 2* which is used to declare a relation's name and its properties. As an extension to SQL can be derived from a uni-sorted or multi-sorted relational model or incorporate ATS a relation can have a number of properties including implicit temporal attributes and be in NFNF. Support for NFNF relations is limited as it can only support one level of nesting for attribute time stamped values because of limitations of MS Access.

4.4 Support for a DML

Once data types have been declared and a test database configured the user then defines additional predicate operators and functions using the DML. A GUI application that is separate from the DDL provides support for the DML and allows the user to define different SQL extensions, their associated operations and functions.

It should be noted that the DML only provides functionality for general queries and cannot be used to UPDATE, DELETE or INSERT data as a data entry screen is provided that performs these functions and, in general, evaluation of query languages focuses on data retrieval. To declare an extension the user first names it and, after doing so, is allowed to define the operators and functions associated with it. This is done in two stages. The user first declares a name for an operator, states whether or not it is a unary or binary operation and the data types that can be used. They have to be declared in the order they are used in the operation with the data type for the left side being declared first. The final stage is the definition of a set of operations that allows the operator to be converted to a set of valid SQL statements of the base RDBMS. An extension can be developed incrementally which permits the user to test the extension as each operator is defined.

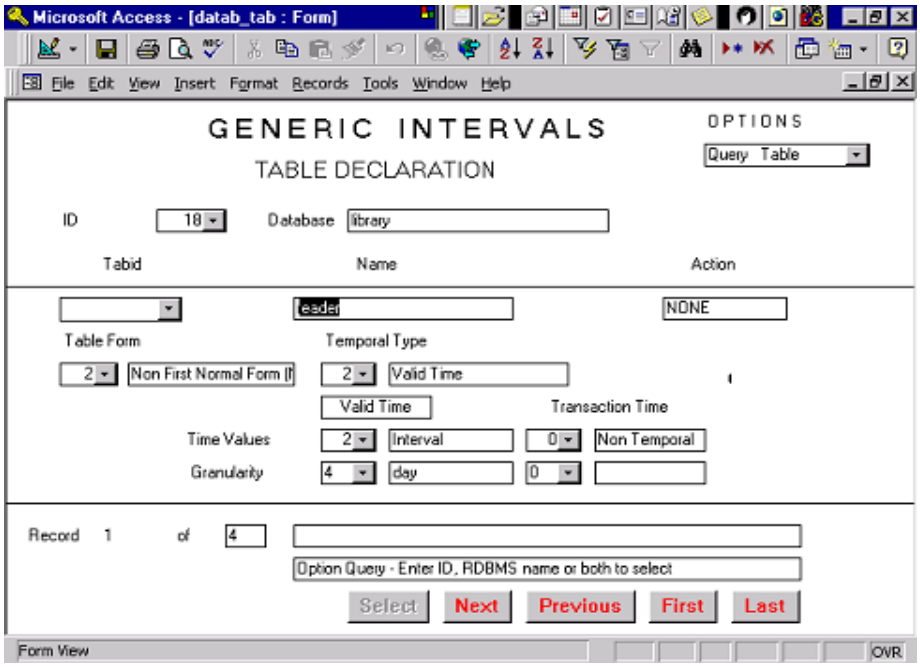

Fig. 2. Data Entry Screen for Base Table Definition

4.5 Definition of Operations

As stated before new data types are declared using a collection of atomic data types that are supported by the underlying RDBMS. When a new data type is declared part of the declaration is the definition of a name that can be appended to the attribute names used in the relation scheme. To define a suitable mapping to the RDBMS' SQL for a non atomic data type's operator the name of the data type's element used on the

left hand side is declared first, followed by a theta predicate and the name of data type's element used on the right hand side. The user also has to declare where in the sequence of mappings the definition occurs and the boolean operation (AND, OR) that follows the mapping. The user continues adding mapping data until the definition is complete.

4.6 Definition of Functions

The base RDBMS used in this application allows functions defined in a database to be included in an SQL statement. This makes the inclusion of functions on data types that do not depend on grouping or involve some other operation on tuples relatively easy to implement. Support for functions or processes like coalescence that form a fundamental part of an extension are more difficult to model and are described later.

To add a function that can be used in an extension's SQL, first define the function in the database supporting the DML. If the function's arguments include a new data type, such as a DATETIME interval, the parameters specified in the actual definition of the base types of which it is made up, are used by the functions' processes.

4.7 Definitions of Functions on Relations

Functions and processes that change the values in the tuple require more consideration. For the temporal and interval extensions such functions are coalescence, FOLD and UNFOLD and NORMALIZE ON. Much of the functionality required to achieve valid time coalescence and FOLD and UNFOLD can be managed using relatively simple processes and the use of temporary tables. (It should be remembered that the prototype is being used to evaluate language extensions not to devise algorithms that improve processing speed so simple procedures are more than sufficient). To see how processes on tuples can be incorporated consider the SQL statements in *Figure 3*.

(A)
```
SEQUENCED VALIDTIME
SELECT A,B,C
FROM TABLEA
```

(B)
```
INSERT INTO XYZ
SELECT * FROM TABLEA
```

(C)
```
SELECT A,B,C,PERIOD1
FROM TABLEA
REFORMAT AS UNFOLD PERIOD1
```

Fig. 3. Queries with a prepended or appended statement to a SELECT .. FROM .. WHERE

Queries A, B and C are similar in that the projection on TABLEA can be executed first and the processing required to achieve the statement SEQUENCED VALIDTIME in (A), INSERT INTO XYZ in (B) or UNFOLD PERIOD1 can be done afterwards. It is evident from (A) that some additions to the projection will have to be made first before the base query is executed and the processes required to achieve coalescence are run. Again these functions, or the elements that make up the users

view of the function, have to be defined first. This information can be held in the system catalogs as illustrated in the tables above.

The inclusion of system catalogs in a conventional RDBMS is well known. Copying that approach has resulted in an RDBMS being used to develop an application that performs basic support for a number of SQL extensions using a single user interface.

4.8 Limitations

To date the majority of work has concentrated on firstly implementing temporal extensions to SQL and secondly implementing extensions to include the generic interval data type. The working application is currently restricted to these data domains although testing of other generic processes continues. It is hoped to develop other generic extensions in the near future as opportunity allows. The authors believe that a substantial range of interesting extensions can be readily modeled although some inherent limitations of MS Access, such as support for only one level of nesting, would probably make a full NFNF impossible.

5. Conclusions

ProSQL is a novel tool for prototyping extensions to SQL. The prototyping tool has been used to define SQL extensions for ATSQL, TSQL and IXSQL in a few days. It has successfully executed queries on databases using attribute time stamping, valid time state tables and schemas using generic interval data types.

Early evaluation of a language or extension is desirable as the results could have a positive influence on the language's evolution. ProSQL allows the user to define an emulation of one or more language extensions without rigidly tying the definition to specific lexical terms. The researcher can experiment with the lexical forms used in a query language extension, perform usability tests, undertake case studies for a range of proposals or use it to evalaute interface characteristics for a proposed extension.

The approach adopted has been to build a wrapper around an existing database management system, in this case Microsoft Access, which allow a designer to define extensions to the basic relational database. The facilities include new data types, new comparison operators as well as temporal features. While there are limitations to the range of language extensions that can be implemented in this way, the authors experience suggests that the range is sufficiently large and flexible to make this approach useful for language developers and HCI researchers.

References

1. Blackwell, A. F. Metacognitive Theories of Visual Programming. Proceedings IEEE Symposium of Visual Languages, 1996, Pages 240-246.

2. Chan, H. C. Wei, K. K, An empirical study on end users' update performance for different abstraction levels. Int. J. Human –Computer Studies (1994) Vol 4, Pages 309-328
3. Davis, J. S, Usability of SQL and menus for database query. Int. J. Man-Machine Studies, 1989, Pages 447-455
4. ORES: Towards The First Generation of Temporal DBMS Valid Time SQL. University Of Athens, Agricultural University of Athens, 1994
5. Goralwalla, I. A., Tansel A. U., Ozsu, M. T, Experimenting with Temporal Relational Databases. CIKM 95, Pages 296 - 303
6. Greene, S. L, Devlin, S.J, Cannata, P. E, Gomez, L.M. No IFS, ANDS, or ORS: A Study of database querying. Int. J. Man-Machine Studies (1990), Vol. 32, Pages 303-326
7. Jarke, M, Turner, J, Stohr, E.A, Vassiliou Y, White, N.H, Michielsen, K. A Field Evaluation of Natural Language for Data Retrieval., IEEE Transactions on Software Engineering, Vol. SE-11, 1985 Pages 97 - 114
8. Lorentzos, N. A, Mitsopoulos, Y. G. SQL Extension for Interval Data., IEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 3, 1997
9. Rumbaugh, J, Blaha, M, Premerlani, W, Eddy, F, Lorensen, W. Object-Oriented Modeling and Design. Prentice Hall International Editions.
10. Reisner, P, Boyce, R.F, Chamberlain, D.D. Human Factors Evaulation of Two Database Query Langugaes- Square and Sequel., Procedings of National Computer Conference (1975), Pages 447-452
11. Reisner, P. Use of Psychological Experimentation as and Aid to Development of a Query Language. IEEE Transactions on Software Engineering, Vol. SE-3, No. 3, 1997, Pages 218-229
12. Human Factors Studies of Database Query Languages: A Survey and Assessment. Computing Surveys, Vol 13, No. 1, 1981
13. Shneiderman, B. Improving Human Factors Aspect of Database Interactions. ACM Transactions on Database Systems, Vol 3., No. 4, 1978, Pages 417-439
14. Snodgrass, R T. The Temporal Query Language TQUEL. ACM Transactions on Database Systems, Vol. 12, No. 2, 1987
15. Stonebraker, M, Brown, P, Object Relational DBMSs – The Next Great Wave. Morgan Kaufman Publishers,
16. Snodgrass, R. T, Bohlen, M. H, Jensen, C. S., Steiner, A. Transitioning Temporal Support in TSQL2 to SQL3. Time Centre Technical Report, TR-9, 1997
17. Toman, D. Point vs. Interval based Query Languages for Temporal Databases. PODS 1996, Pages 58 - 67
18. Torp, K, Jensen, C. S, Snodgrass, R. T, Stratum Approaches to Temporal DBM Implementation, IDEAS Cardiff, 1998
19. Welty, C, Stemple, D.W, Human Factors Comparison of Procedural and Non Procedural Query Language. ACM Transactions. on Database Systems Vol. 6, No.4, 1981 Pages 626-649
20. van Welie, M, van der Veer, G, Eliens, A. Breaking Down Usability. Human Computer Interaction – INTERACT 99, 1999, Pages 613 - 620
21. Yi-Miin Yen, M, Scammel, R. W. A Human Factors Experimental Comparison of SQL and QBE. IEEE Transactions on Software Engineering, Vol. 19, No. 4, 1993, Pages 390 – 409