Technical Report

CMU/SEI-87-TR-48
ESD-TR-87-211

Interfacing Ada and SQL

Charles Engle
Robert Firth
Marc H. Graham
William G. Wood
December 1987

Technical Report

CMU/SEI-87-TR-48
ESD-TR-87-211
December 1987

Interfacing Ada and SQL

Charles Engle
Robert Firth
Marc H. Graham
William G. Wood

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler SIGNATURE ON FILE
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Interfacing Ada and SQL

Abstract: The Software Engineering Institute was asked by the Ada Joint Program Office to investigate the problem of interfacing programs written in Ada with Database Management Systems (DBMS) implementing the Structured Query Language (SQL) database language. The authors decided to concentrate on a description of the problems involved in producing an interface which would be worthy of becoming a standard. This document is meant to assist the reader in answering the question "What constitutes a good interface between Ada and SQL?" The document should be useful both in the production of a standard and in the analysis of any proposed standard.

1. Introduction

The programming language Ada and the database language SQL each represent the culmination of decades of academic and industrial research in their respective problem domains. Their user communities, in particular the Department of Defense, are now faced with the task of devising a standard interface between the two systems that will facilitate the construction of Ada application programs that access data stored by an SQL DBMS. In this context, the Ada Joint Program Office requested the assistance of the Software Engineering Institute. The authors of this report spent approximately two months reading and discussing the available literature. (See the references section for a partial listing.) We decided to concentrate on a description of the problem, rather than any proposed or new solution. This report should be useful both in the production of a standard and in the analysis of any proposed standard.

2. The Problem

The problem requires specifying a standard interface which permits an application program written in Ada to access and manipulate data controlled by a Database Management System that responds to directives written in the database language SQL.

This formulation of the problem statement makes a point by omission. The authors feel that the problem of interfacing Ada and SQL should not become intertwined with the problem of devising an ideal persistent data storage and retrieval mechanism for Ada. Rather, the authors feel that Ada and SQL should each be used as is and that the requirements and design goals of each should be given equal weight in the standard.

3. Constraints on the Solution

Certain economic factors put constraints on the standard. These constraints are:

- **Portability**. A primary purpose of standards promulgation is the provision of portability for user-written software. The Ada-SQL interface should support the portability—among target hardware, operating systems, Ada compilers and SQL engines—of all programmer-written code.
- **Separate sourcing**. It must be possible to acquire the Ada compiling system and the SQL engine from separate vendors. In the current marketplace, different software companies provide Ada compilers and SQL engines. Although the large hardware vendors offer both in their product lines, the model of separate sources is a realistic one.
- **Shared data - Interoperability**. It must be possible for the database to be accessible from and updatable by non-Ada programs. These programs include the utilities, such as forms interfaces and data entry utilities, provided by the DBMS vendor. Furthermore, there are multi-language shops which have a large investment in non-Ada SQL applications that they will not be willing to discard.
- **Efficiency**. The solution should not impose excessive compile or run time costs. In particular, at run time it is reasonable to suppose that the SQL interface will be fairly deeply nested within the loop structure of the application program, making optimization a critical issue. The standard must avoid mandating any feature for which no efficient implementation exists.
- **Standard preservation**. The solution should avoid modifications to either the Ada or the SQL standard. Considerable effort has been expended on each of them. Their interface should not require them to be modified.
- **Range of acceptance**. The solution must be acceptable to a large segment of the user community. Should the Department of Defense mandate an interface which is rejected by the wider community, it will lose many of the benefits of standardization. Vendors will be reluctant to produce such an interface, thereby diminishing competition. Programmers will be less likely to be trained in the interface, thereby increasing training costs.

4. The Issues

This section presents a list of the technical issues confronting the designer of any interface between Ada and SQL. Although they are presented as a list, the issues are interrelated. Techniques adopted to solve one of the issues will restrict and affect techniques available to solve others. We have not attempted to make these interrelationships explicit.

Further, we have restricted our presentation to high level issues. The choice of a strategy for dealing with any issue raises myriad issues of implementation. We feel it inappropriate to include such secondary issues here.

Programming Issues

4.1. Typing System Differences

In this section we detail narrow, technical problems arising from the differences in the typing systems presented by the SQL and Ada languages. Proper solutions to these problems are vital to the overall quality of the interface.

1. **Basic, atomic or scalar types.** The core of any interface between SQL and Ada is the type mapping. This mapping assigns, to each SQL data type, a corresponding Ada type. The issue is what this mapping should be. A type mapping is said to be *direct* if it assigns to SQL type *T* the Ada type *U* with the following properties:

   - the value sets of *T* and *U* are identical and
   - each bit pattern representing an abstract value of the SQL type *T* represents the same abstract value as a bit pattern of the Ada type *U*.

   A direct type mapping ensures that the selected Ada type correctly represents the SQL type without loss of information and without run time data conversion. The discovery of a direct type mapping is impeded by both standards' leaving as implementor decisions the choice of value sets underlying a given type. It is therefore possible for the Ada compiler and SQL DBMS to give different semantics, in the sense of value sets, to similar appearing types (e.g., INTEGER).

   Direct type mappings are further hindered in the heterogeneous case. In this case the Ada application and SQL DBMS run on separate machines with differing encoding schemes. Direct mappings are impossible in this case. It may be appropriate for the Ada-SQL interface standard to assume that the heterogeneous representation problem is addressed by other software.

   There are some Ada scalar types which do not appear to have direct analogues in SQL. These are enumeration types, **range** constrained numeric types and fixed point types. (SQL allows only decimal fixed point types.) The issue is whether and how values of these types should be stored in SQL databases.

2. **Non-elementary types**. Ada's type formers are **record, array** and **access**. Should values of any or all of these types be storable in SQL databases? Conversely, SQL has the implicit type formers tuple and relation. Should values of these types be definable and accessible in Ada programs?

3. **Operations**. A type mapping must do more than map SQL types to Ada types (and conversely). It must also indicate how the arithmetic and logical operations of SQL are reflected in Ada (and conversely). Among the most difficult problems in this area is the varying treatment of fixed point types. SQL treats these as exact numeric types; Ada treats them as approximate. The difference manifests itself in fixed point multiplication. (SQL leaves the scale of fixed point division results implementor-defined.) The interface must decide what to do about the possibility that the comparison A=B $\times$ C may get different answers from the two systems. Similar comments apply to the comparison of string values of different lengths. In Ada, these are never equal, whereas in SQL they may be.

   The interface must also decide what to do about the varying strengths of the typing models of SQL and Ada. SQL allows boolean and arithmetic expressions to be formed using values of any arithmetic types. Ada insists that the operator be expressly defined for the types of the expression. An expression which is legal in SQL may well be illegal in Ada unless the interface makes provision for allowing it. The issue then arises as to whether such provision is desirable.

4. **Null values and three valued logic**. SQL columns may, under administrative control, have a specialized value called "null". This value has particular semantics for capturing partial information. For example, a comparison of the null value to any other value, including null, results in a third truth value, called "unknown." It seems that any Ada-SQL interface must provide the programmer with a means of determining if a value is null. The issue is whether or not the interface prevents the programmer from inadvertently using a null value as though it were not null. Such use will cause incorrect results to be calculated without any indication or exception being raised.

4.2. Defining the Types of Parameters

Each parameter passing across the Ada-SQL interface must have a definition visible to the Ada compiler and one visible to the SQL engine. Should these definitions be encoded separately, and thus potentially incompatibly, or is it possible for the interface to derive one of the definitions from the other, thereby saving effort and avoiding error?

Can the Ada derived type mechanism be used fruitfully in the interface? For example, is it possible for the interface to allow the Ada programmer to treat an SQL column value of SQL type INTEGER as an Ada derived type EMPLOYEE_NUMBER? This prevents employee numbers from being compared to part numbers. Is it possible for the subtype mechanism to be used in the derivation of EMPLOYEE_NUMBERS_IN_DIVISION_X, for example? Can private and private limited types be used to prevent arithmetic operations on employee numbers, if that should be the user's choice? If these facilities are desirable, how can they be implemented? In particular, how can an implementation gather the information needed to use the Ada typing mechanisms correctly?

4.3. Exception Handling

ANSI standard SQL signals exceptional conditions to the application program by means of the ubiquitous parameter SQLCODE. Ada incorporates an exception handling mechanism for dealing with exceptional conditions. Should the Ada interface to SQL accept the ANSI proposal, or should it be required to raise appropriate exceptions? Should the names of these exceptions be part of the interface standard?

Architectural Issues

4.4. Program Text Preparation

Should an Ada-SQL application be prepared as a single piece of text containing directives to both the Ada compiler and the SQL engine? This question should be considered in light of the typing system differences cited above and also the differences in the underlying paradigms. Ada programs are procedural specifications which process one thing at a time. SQL statements are declarative specifications which operate on the database as a whole.

The three primary proposals for the Ada-SQL interface can be distinguished by their solutions to this issue. The "SQL in Ada" [6] and "embedded" [1, 3] approaches are similar in that they each require a single program text having both Ada and SQL directives. SQL in Ada solves the problem by having the SQL code interpreted by the Ada compiler. Typing system differences are handled by using Ada typing everywhere. Conceptual differences remain. The embedded approach, requiring a preprocessor, leaves the typing system and conceptual differences unresolved.

The third approach, known variously as the "module" [5] or "view processor" [9] approach, stipulates distinct text streams for Ada and SQL. The differences are not resolved solely by separating the text streams. However, the problem is not compounded by intermingling the texts. The text separation seems to introduce a new problem of configuration management: that is, ensuring that the relationship of the two texts is known and controlled.

4.5. Recompilation and Schema Evolution

As a database evolves, its schema, the set of governing definitions, is modified to reflect the changing needs of the organization. Preexisting application programs may or may not be affected by these changes. It may be that the changes do not affect data used by a given program. It may be that the program's view, its preconception of the structure and semantics of the data, can be reconstructed from the new database. Database management systems provide facilities for preserving the correctness of programs at the object level. An Ada interface can be judged on the degree to which it permits or denies such "data independence."

Not all schema modifications permit application program correctness to be preserved. The Ada library system records, with respect to Ada program modifications, information regarding subsequent modifications which may be needed. Is it possible for the Ada SQL interface to signal in some way the identity of Ada compilation units which will require at least recompilation if not rewriting as a result of a schema change?

4.6. Query Compilation

Relational DBMSs differ from most other DBMSs in that relational queries allow a greater degree of optimization. The optimization process is commonly called "query compilation" and it consumes a significant amount of time. In high performance transaction environments, certainly, this time is not available during transaction execution. Can an Ada-SQL interface allow for query optimization at compile time? An interface that does not allow such precompilation may not be suitable for high performance applications.

Implementation Issues

4.7. Difficulty of Interface Implementation

The implementation of an Ada compiler or an SQL DBMS is an enormous task. In contrast, the implementation of the Ada-SQL interface should be relatively easy. If the interface requires a prohibitively expensive implementation effort, vendors may not be willing to undertake its production.

4.8. Uniformity

The ANSI standard for embedding SQL into programming languages is remarkable in that the embedding proceeds in a uniform way for all languages. This makes the implementors' task easier in that much of the preprocessor which makes up the implementation can be reused from language to language. The ANSI Ada interface therefore resembles the FORTRAN interface. Does this adequately serve the needs of the Ada community?

4.9. Independence from Targets

Is it possible for the interface software to be written in such a way as to be independent of either the actual Ada compiler or SQL DBMS in use at a given site? If this is not possible in absolute terms, is it possible to restrict this dependence to localized areas of the software? These questions should be answered in light of the probable suppliers of Ada-SQL interface software, who might be Ada suppliers, SQL suppliers or third parties.

4.10. Range of Applicability

Can the Ada-SQL interface be used in the full range of development organizations and target environments? These include:

- the corporate information center (mainframe)
- the departmental environment (mini)
- the workstation or personal computer (micro)
- real-time embedded systems

Although efficiency is a concern, a paramount issue is the presence of a database administration (DBA) function. Will the interface require a DBA? Will it be able to exploit a DBA?

4.11. Validation

The validation of Ada compilers gives Ada users a high degree of confidence that their compilers actually compile Ada code correctly. An ironclad guarantee of correctness is difficult to produce in practice. Can the benefits of validation be extended to Ada-SQL users? Will such validation require the validation of SQL engines, a task which would seem quite formidable? Can the interface be meaningfully validated without a validation of the DBMS? Against what correctness criteria can such validation be done? Who can and should do such validation?

5. Conclusions

In this report, we have delineated the problems involved in interfacing Ada application programs to SQL database management systems. We hope to have provided the reader with sufficient information to form his or her own opinion of a proposed Ada-SQL interface. However, we would not seek to deny that we have formed opinions of the desirable characteristics of such an interface. We discuss 9 of these characteristics in this section. The reader should be advised that we do not have an interface to propose nor do we have any direct, concrete evidence that there exists an interface supplying all the features we would like to see.

1. On the issue of basic types, we feel that the demands of interoperability mandate that only values of SQL data types be stored in an SQL database. If non-Ada tools are to operate on the data consistently, then they must be able to interpret the data in the same way as the Ada tools do. This does not prevent installations which can prohibit such non-Ada tools from making local decisions to ignore demands of interoperability.
2. With respect to the differences in the syntax and semantics of the logical and arithmetic operators, we note that in any interface, the SQL query will eventually be processed by the SQL engine, which will process it using SQL rules. We feel the rules of Ada should apply to Ada code and the rules of SQL to SQL code.
3. We feel the interface must prevent inadvertent use of null values, as our experience indicates such use to be a significant source of errors in database applications.
4. We would very much like to see an interface that allows a parameter type to be defined only once. We would like an interface which permits the Ada programmer the use of derived and subtyping and limited private typing as appropriate to the situation.
5. We would like the Ada exception mechanism to be used in handling exceptional SQL conditions.
6. In the matter of program text preparation, we favor the separate text stream model, although we admit this introduces new problems of configuration management. Our conclusions in 2 suggest that a single text containing both SQL and Ada introduces confusion, as the syntax and semantics of that text would vary from line to line.

7. The interface must not weaken the ability of the DBMS to provide data independence to application programs at the object level. It is unclear the extent to which the SQL data dictionary and Ada library can be made to cooperate in the identification of needed changes.
8. The interface must allow compile time query optimization for it to be useful in applications with run time performance constraints.
9. We would like to see a validation process for the Ada-SQL interface software, although we do not have proposals for the solution of the technical and organizational problems involved.

References

[1] *American National Standard for Information Systems - Database Language - SQL X3.125-1986.*
American National Standards Institute, 1986.

[2] *Procedure Language Access to Draft Proposed American National Standard Database Language SQL.*
American National Standards Institute X3H2, 1985.

[3] *(draft proposed) Database Language Embedded SQL.*
American National Standards Institute X3H2, 1986.

[4] *(working draft) Database Language SQL Addendum-2*
American National Standards Institute X3H2, 1986.

[5] Boyd, S.
SQL and Ada: The SQL Module Option.
Technical Report, COMPASS, Wakefield, Ma, 1987.

[6] Brykczynski, Bill R., Friedman, Fred.
Preliminary Version: Ada/SQL: A Standard, Portable, Ada-DBMS Interface.
Technical Report P-1944, Institute for Defense Analyses, Alexandria, Va., July, 1986.

[7] Brykczynski, B.
Methods of Binding Ada to SQL: A General Discussion.
Technical Report, Institute for Defense Analyses, Alexandria, Va., 1987.

[8] Donaho, J.E.D., Davis, G.K.
Ada-Embedded SQL: the Options.
Ada Letters VII(3):60-72, May, June, 1987.

[9] Smith, J. M., Chan, A., Danberg, S., Fox, S., Nori, A.
A Tool Kit for Database Programming in Ada.
Technical Report, Computer Corporation of America, Cambridge, Ma., 1984.

Table of Contents