Proceedings of CASCON'1991, pp 205-222, Toronto, Canada, 1991

Supporting Database Access in the Hermes Programming Language*

P.-Å. Larson† Qiang Zhu† Frank Pellow‡

Abstract

The work reported in this paper is part of a project aimed at designing and prototyping an application development environment that allows easy development of platform-independent distributed applications. The main goals of the database subproject are to investigate methods for (1) providing (SQL) database access and (2) supporting transaction management within a distributed programming environment based on the paradigm of communicating sequential processes. This paper looks at how SQL database access can be provided in Hermes, a new language for distributed programming based on this paradigm. The paper compares the characteristics of Hermes and SQL tables, discusses potential ways of supporting database access in Hermes, and then defines an embedding of SQL in Hermes. Some implementation aspects are also discussed.

Keywords: distributed applications, database access, Hermes, SQL.

*The research reported here was supported by IBM Canada Ltd. This paper is also available as IBM Canada Laboratory Technical Report TR 74.068

†Department of Computer Science, University of Waterloo, Ontario, Canada.

‡Centre for Advanced Studies, IBM Canada Laboratory, Toronto, Ontario, Canada.

1 Introduction

As distributed computing systems emerge, increasingly complex and sophisticated applications will be required. These applications will consist of multiple communicating processes distributed over a large, constantly evolving, heterogeneous network of computers. Developing such applications using today's technology is feasible but difficult and expensive. The basic problem is caused by working at too low a level of abstraction; having to master too many technical details about the underlying platform (hardware, network, protocols, and so on). Hence, applications become platform-dependent which (as we know from bitter experience) ends up being very expensive. To remove platform dependence, the level of abstraction at which applications are developed must be raised.

The main objective of the CORDS project is to design and prototype an application development environment that allows easy development of platform-independent distributed applications. The programming model adopted by the project, hereafter referred to as the *process model*, is based on communicating, sequential processes. In this model, an application is structured as a collection of concurrently executing processes that communicate by message passing. The application developer's task is to design and implement these processes. The mapping of these processes onto the currently

existing platform should be a completely separate concern, and should not be allowed to influence the application logic.

Virtually all applications require access to shared, persistent data. Frequently, such data is stored in and managed by some database system. It would be naive to assume that all data is managed by a single (even if distributed) database system; more typical would be a scenario where it is managed by multiple, heterogeneous database (and file) systems. One important objective is to provide application programmers with a simple and consistent view of database access; a view that is completely independent of how and where the data is stored. Mapping this view onto a physical reality consisting of multiple, heterogeneous database systems running in a distributed environment is the main challenge of the database subproject.

Hermes is a new programming language [10]. It based on the process model and is specifically intended to support distributed computing. One aim of the CORDS project is to evaluate the suitability of Hermes for the development of distributed applications. Given the widespread use of relational databases and SQL, it is imperative that support for database access through SQL be added to Hermes. Hermes provides a table data type and several built-in table operations. From a database point of view, this is one of its most interesting features. It raises the question whether Hermes and SQL tables can be integrated, that is, whether SQL operations can be applied to Hermes tables and Hermes table operations to SQL tables.

This paper proposes a way of adding support for database access through SQL to Hermes. Section two contains a comparison of the characteristics of Hermes and SQL tables, including table operations. In section three, we discuss potential models for supporting database access. Section four describes our proposed solution. A few implementation considerations are discussed in section five. The last section provides a summary and a list of issues requiring further investigation. A syntax description is contained in an appendix.

This paper concentrates on language-level integration on SQL and Hermes; many other issues are ignored. Work on transaction management is in progress and will be discussed in a separate paper [6].

It is assumed that readers of this paper is familiar (at least at the introductory level) with both SQL and Hermes. There are many variants of SQL. Unless otherwise stated, the assumed variant is the one defined by ISO 9075-1989 [1], hereafter referred to as SQL-89.

2 Comparison of Hermes and SQL Tables

2.1 Table Characteristics

A Hermes table is a collection of values of the same type and typestate. A table may be *ordered* or *unordered*, and may have one or more *keys*. A key consists of one or more fields. Any two elements of the table will have different values of all keys. A component in a Hermes table may itself be a table, resulting in a *nested table*. A table where all components are atomic (single-valued, unary) will here be referred to as a *flat table*.

One notable characteristic of Hermes is its lack of global data; every variable belongs to a process and cannot be referenced (directly) by other processes. At first glance, this seems to imply that Hermes processes cannot share data. However, a Hermes process may "export" functions, called *ports*, that provide access to its data. (This is similar to the concept of access functions or operations for abstract data types.) Any process with the necessary capabilities can invoke an exported function by sending the owner process an appropriate mes-

sage. The net effect is that all operations on shared data are controlled by the owner process. Since Hermes supports persistent processes, a Hermes table within a process can be used for storing permanent data.

An **SQL table** is a multi-set of rows, where each row has the same cardinality and contains a value for every column of the table. A row is the smallest unit of data that can be inserted into or deleted from a table. The value of a column in a row is the smallest unit of data that can be updated or selected from a table. A value is either a *null value* or a *non-null value.* An SQL table may have *constraints* associated with it. Three types of constraints are allowed: *unique (key)* constraints, *referential (foreign key)* constraints, and *check* constraints.

Comparing the characteristics of these two table types, we see that:

- An unordered, flat Hermes table is equivalent to an SQL table. (In Hermes, character strings are defined as ordered tables of characters; even so, we still consider a Hermes table containing character strings to be flat.) However, the table data type in Hermes is broader than SQL tables and includes nested tables and extendible arrays (ordered tables of unlimited size). Fig 1 illustrates the characteristics that can be specified for a Hermes table and the subset corresponding to SQL tables.

- Keys for Hermes tables play the same role that unique constraints do for SQL tables. Hermes has no direct equivalent to SQL's referential constraints and check constraints.

- An uninitialized component of a tuple in a Hermes table may be considered to have the (SQL) value null. It is not clear at present whether the semantics of (SQL) null values can be implemented by Hermes' typestate mechanism.

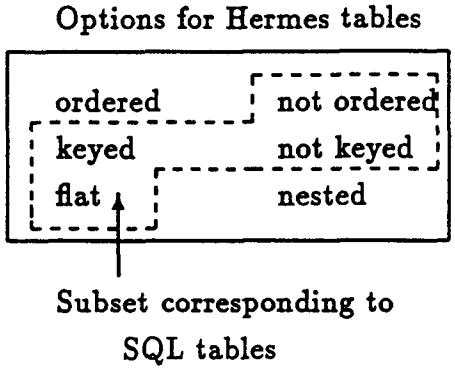

Figure 1: Hermes and SQL table characteristics.

- SQL provides authorization and access control features. There are no direct equivalents in Hermes.

- SQL supports viewed tables that may be updatable, possibly with a check option. There is no direct equivalent in Hermes.

2.2 Table Operations

Both Hermes and SQL provide table (set) operations. Some operations are available in both languages, but in general SQL provides more powerful ones.

- Hermes provides the following set-level operations: **every of** (makes a copy of a subset), **insert**, **extract**, **remove**, **merge** (union). The **inspect** statement makes a copy of a selected element available, and the **for-inspect** statement iterates over selected elements. The Boolean functions **exists** and **for all** test whether at least one or all elements, respectively, satisfy a given condition.

- No update operation is provided in Hermes. An update operation can be implemented by using remove and insert operations.

- A Hermes retrieve operation cannot directly join data from two or more tables. A join would have to be implemented by nested selectors, as illustrated in the following example. (π stands for projection, $\bowtie$ for join, and σ for selection.)

$\pi_{R1.*}(R1 \bowtie_{R1.a=R2.a} R2)$
is equivalent to
every of $t1$ in $R1$
where (exists of $t2$ in $R2$
where ($t1.a = t2.a$)) .

- Unlike SQL, Hermes does not combine projection with its retrieval operations. For example, we cannot implement $\pi_{R1.a,R1.b}(\sigma_{R1.a>10}(R1))$ by using Hermes every of alone if $R1$ has other columns in addition to a and b. Separate assignment statements are required to implement a projection.

- It appears that all Hermes table operations can be easily simulated by SQL table operations. The reverse must also hold because Hermes has the power of a general-purpose programming language.

- Several facilities provided by SQL are not (directly) available in Hermes, for example, (1) set functions (AVG, MAX, MIN and SUM); (2) additional predicates (LIKE and IS NULL); (3) some clauses in a query (GROUP BY, HAVING, ORDER BY); (4) transaction management (COMMIT WORK, ROLLBACK WORK).

3 Potential Models

The reasons for extending Hermes with facilities for database access through SQL are straightforward:

- It provides access to the large amounts of data stored in external databases.

- It enables applications to exploit the services provided by database systems.

- It enables application programs written in Hermes to share (persistent) data with applications written in other languages.

- Relational databases and SQL are widely used and have become standard "tools" for application development.

We see two different conceptual models for adding database support to Hermes, which we will call the database programming language (DBPL) model and the database server model.

The **DBPL model** advocates a complete integration of database and programming language facilities. If a relational model is used, this amounts to adding the notion of global, persistent tables to some base language. To achieve complete integration, it must be possible to manipulate such tables using normal programming language constructs. In essence, the programmer's view is one of "importing" a table (by some declarative statement) into his program, after which it can be manipulated using normal facilities in the base language.

Hermes already supports tables and table operations. If we add the concept of global tables, database access can be provided by mapping database tables onto Hermes tables. The basic idea is simply to provide some way of declaring that a table is an imported database table (or view). (We ignore the possibility of a Hermes process "exporting" a table.) The following question arises immediately: What operations are allowed on imported database tables,

that is, Hermes table operations, SQL table operations, or both?

Allowing only Hermes table operations is appealing because of conceptual simplicity: all that is added to the language is the notion of importing a database table. However, Hermes table operations are more restricted than SQL table operations. Consequently, programmers would have to

- write more code and
- repatedly redevelop and re-implement operations already provided by database systems.

SQL provides powerful table operations and the system takes responsibility for finding the best way of executing operations requested by users. This is one of the main advantages of relational database systems. By not supporting SQL operations, Hermes programmers would not be able to take advantage of these services. Disallowing SQL operations does not appear to be an acceptable solution.

Allowing SQL operations to be applied to all tables would require extensions to SQL. As discussed in section two, SQL-89 cannot handle ordered tables or nested tables. Extensions to SQL for such tables have been proposed [7, 8, 9]. However, they are not supported by any widely available, mature database system.

The database server model retains a strict separation between data internal to a program and external data managed by a database system. The programmer's view is one of interacting with a server that provides certain services, in particular, services for retrieving, storing and updating external data. They are related in a client-server relationship.

This model is easier to implement, mainly because of the looser integration with the programming language. All that needs to be defined is the client-server interface, that is, how to request database services in an application program and how to pass data and status information between the client and server. In most practical implementations, this interface is defined at two levels: a call-level interface and programming language embeddings. The call-level interface consists of a number of function calls and associated parameters and is intended to be language independent. To make application programming easier (less tedious, less error-prone), an embedded interface can be provided. This interface allows SQL statements to be embedded in the programming language. A preprocessor is used to extract these statements and translate them into "native" code, including calls to the server. Existing (and proposed) SQL standards are all based on the database server model [1, 2, 4, 5], attempting to standardize both server functionality (SQL statements) and the embedding of SQL statements in a variety of programming languages.

For this project, we have decided to use the database server model and to define an embedding of SQL in Hermes. This provides the required functionality and has several advantages:

- It provides a clear separation of database issues and facilities from language issues and facilities. Experience has shown such "separation of concerns" to be highly desirable.
- It is consistent with the model used in existing standards and SQL embeddings in other programming languages.
- The looser integration makes it easier to accommodate variants and future extensions of SQL and/or Hermes.
- It is significantly easier to implement than attempting to extend Hermes into a database programming language.

4 Embedding SQL in Hermes

Current SQL standards [1,2], define two methods for providing a program with database access. In the **module method**, the application program is divided into two separate modules: a host program and an SQL module. All SQL statements are placed in the SQL module; there are no SQL statements in the host program. All database access requests are issued through procedure calls in the host program (in Hermes, they would be outport calls). The host program can be compiled in the normal way while a special SQL module compiler would be used to compile the SQL module. The resulting object modules are then linked in the normal way. This provides a clean separation between the host language and SQL.

In the **embedded method**, SQL statements and host language statements are intermixed in the same source program. Before the program is compiled, it is run through an SQL preprocessor that extracts all SQL statements and replaces them with function calls (and possibly other statements in the host language). Exactly what happens to the extracted SQL statements varies. Some preprocessors do little more than validate the syntax and convert each statement to a text string which is then used as a parameter in a call to the database system. Note that this solution does not require any communication with the database system at compile time and that the target tables need not even exist yet. At the other end of the spectrum, we have preprocessors that perform complete syntactic and semantic analysis, optimization, and compilation into an access plan at this stage. (Much of this work is in fact done by the database system.) The preprocessor must be able to interact with the database system at compile time and the target tables must have been defined.

The embedded method is, by far, the more popular one and we have chosen this method as well. A Hermes/SQL program is a compilation unit that consists of Hermes text and SQL text. The Hermes text must conform to the Hermes Reference Manual [10]. The SQL text must conform to the syntax described in an appendix of this paper. Compared with SQL-89, our version of SQL has the following special properties:

- There are no cursor-related statements (OPEN, FETCH, CLOSE). The main use of SQL cursors is for fetching the result of a query one row at a time. This is required when SQL is embedded in a language that does not have a set or table type. In Hermes, the result of a query can be loaded directly into a Hermes table. Hermes tables are (theoretically) of unlimited size. If needed, normal Hermes features can then be used to process the result.
- As a consequence of the above, SQL-89's Positioned UPDATE and Positioned DELETE statements are not allowed. (These statements either modify or delete a row pointed to by a cursor.) This is not a serious loss. The same effect can be achieved by Searched UPDATE and DELETE statements on any table that is defined with a primary key (and we argue that all tables should be so defined).
- An INSERT statement can insert data directly from a Hermes table or record. A SELECT statement can retrieve data into a Hermes table, a Hermes record, or a list of Hermes unary variables. Thus data transfer between internal Hermes tables/variables and external SQL tables is straightforward.
- We have chosen to use SQLSTATE (from SQL2) for returning errors and warnings rather than SQLCODE in SQL-89.

- Several redundant phrases, for example, EXEC SQL, BEGIN DECLARE SECTION, END DECLARE SECTION, are optional. Omitting these redundant phrases results in a "cleaner" language. However, for reasons of compatibility with SQL-89 standards for embedded SQL, we decided not prohibit the use of these phrases.

- Like most practical SQL implementations, we will extend SQL-89 by allowing some schema statements to be embedded. Schema statements tend to vary a great deal among different SQL implementations. We intend to support many different underlying database systems and need be able to handle requests that contain product-unique features. We would like to identify a basic set of schema statements that (1) meets users' basic requirements for defining, manipulating, and controlling access to tables and views; (2) can be easily supported by all the database systems that we utilize; (3) conforms to ISO SQL standards. SQL-89 schema facilities are not sufficient (for instance, they do not provide statements to add columns to a table or to drop a table). In the appendix, we have tentatively included CREATE TABLE/VIEW, ALTER TABLE, DROP TABLE/VIEW, and GRANT/REVOKE statements. We show the SQL2 syntax of these statements.

- It appears that dynamic SQL can be supported without circumventing Hermes type system and type checking. In Hermes, programs can not only be executed but also manipulated as values (of type program). We can create, modify, compile, and run a program from within a Hermes process. It should be possible to use this feature to handle dynamic SQL statements by creating, on the fly, a Hermes program with embedded SQL statements. In fact, this may allow us to generalize and extend dynamic SQL. Dynamic Hermes/SQL programs will be discussed in a separate paper.

A Hermes program with embedded SQL consists of a declaration part and a body part. In the declaration part, every Hermes host variable that SQL uses to interact with the Hermes host program must be declared, either explicitly (using <embedded SQL begin declare> and <embedded SQL end declare>) or implicitly (without using the phrases). A program body part contains:

- an <embedded exception declaration> which specifies the action to be taken when an <SQL statement> causes an exception condition;

- a <connect statement> which establishes a connection to a database;

- one or more <SQL statements>. This includes <data statement>, < schema statement>, <transaction statement>, and <dynamic statement> expressions.

The following example illustrates the proposed embedding of SQL in Hermes. The example uses a database DB1 with three tables: UNDERGRADUATE, GRADUATE and OLDSTUDENT, each one with four columns: NO (INTEGER), NAME (CHAR(20)), AGE (INTEGER) and STATUS (INTEGER). We want to retrieve all the rows for older students (AGE>=40), do something with them, then insert them into the OLDSTUDENT table. The following (incomplete) program shows how to accomplish this.

```
1  EmbeddedProgram:  using( ... )
2  process( ... )
3   declare
4    Newstatus : integer ;
5       ......
6    EXEC SQL BEGIN DECLARE SECTION;
7      Result :   TableType ;
8      Oldage :   integer ;
9      SQLSTATE : SqlStateType ;
10   EXEC SQL END DECLARE SECTION;
11      ......
12  begin
13    EXEC SQL
14    WHENEVER SQLERROR RAISE :Errcheck;

15    EXEC SQL  CONNECT DB1 ;
16    Newstatus := 15 ;
17    Oldage := 40 ;
18      ......
19    EXEC SQL  SELECT  *
20              INTO TABLE  :Result
21              FROM  UNDERGRADUATE
22              WHERE  AGE >= :Oldage
23              UNION
24              SELECT *
25              FROM GRADUATE
26              WHERE  AGE >= :Oldage ;
27      ......
28    for y in Result where ('true')
29    inspect
30      y.status := Newstatus ;
31      ......
32      EXEC SQL INSERT INTO OLDSTUDENT
33               RECORD :y ;
34    end for ;
35      ......
36    on ( Errcheck )
37      ......
38 end process
```

Lines 6-10 specify the interacting Hermes variables and SQLSTATE. Lines 13-14 say that when an SQL statement causes an error, the exception Errcheck is to be raised. Lines 19-26 retrieve the required data from UNDERGRADUATE and GRADUATE into the Hermes table Result. Lines 28-34 use normal Hermes control statements to iterate over the records in Result. Lines 32-33 insert the modified record in y into the SQL table OLDSTUDENT. Line 36 begins the exception handling routine for Errcheck.

Instead of inserting the records one by one into OLDSTUDENT, we could first have modified the records in Result (by Hermes remove and insert operations) and then issued a single SQL statement: `INSERT INTO OLDSTUDENT TABLE :Result ;`

5 Implementation Considerations

A Hermes/SQL compilation unit (program) defines one or more processes. Four steps are required in order to execute a Hermes/SQL program:

1. preprocessing,
2. (Hermes) compilation,
3. linking and loading, and
4. running.

The SQL preprocessor extracts all SQL statements from the source program and produces a "pure" Hermes program. Every executable SQL statement is replaced with an outport call to a "shadow" process. Each Hermes process in the program has an associated shadow process, which is created at the same time as the "owner" process. The shadow process implements the SQL statement(s) extracted from the owner process by communicating with the database system. Figure 2 illustrates the runtime structure.

Some mechanism is needed for reporting errors, warnings, status, and statistics from servers back to application programs. A Diagnostic Management facility is being planned. According to current plans, it will consists of two parts: (1) facilities for servers to post information; (2) facilities for application programs to retrieve information. The facility will be make use of SQL2's SQLSTATE code (a return code with different ranges for diagnostics identified by the standard, by individual products and by user-written applications). It will be similar to SQL2's Diagnostic Management scheme and provide mechanism for reporting several warnings/errors raised by a single request from an application program). Although it will be based on

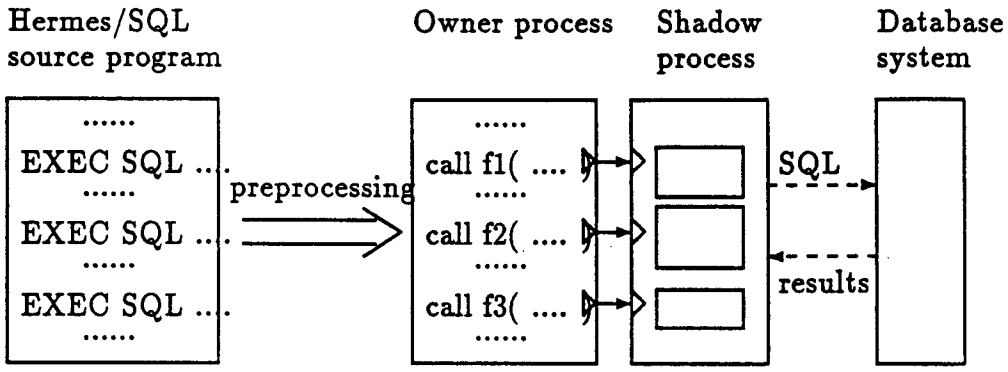

Figure 2: Runtime structure.

these SQL2 facilities, it will differ in many particulars and have a broader scope.

The SQL tables referenced by an SQL request might not all be managed by the same database system. If so, we need to decompose the request into "smaller" single-database request, and perform whatever processing is necessary to combine the results obtained from individual databases. Optimal request processing in an environment of multiple heterogeneous database systems is one of our main research topics.

6 Summary

Hermes' concept of a table is more general than that of SQL. In particular, nested tables and ordered tables are supported by Hermes but not by SQL. However, the table operations provided by SQL are much more powerful than those provided by Hermes.

Users need to be able to access data in external (relational) databases from Hermes programs. Two models for providing this capability were outlined: the database programming language (DBPL) model and the server model. The DBPL model attempts a complete integration of the programming language and database services and strives to eliminate the distinction between internal and external data. The database server model is less ambitious and retains a strict separation between internal and external data. All that is provided are means for requesting services and interchanging data between the server and application programs.

We decided to adopt the database server model. It provides the required functionality, provides a clean separation, and is significantly easier to implement. An application program requests database services by SQL statements embedded in the source program. The syntax for SQL embedded in Hermes has been designed and an SQL preprocessor for Hermes will be implemented. For each process in the user's program, the preprocessor will generate a "shadow " process which handles all interaction with the underlying database system(s). A process interacts with its shadow process through normal calls.

The following issues are not fully discussed in this paper, and will require further investigation:

- implementation of null values in Hermes;
- interface between an "owner" process and its shadow process;
- interface between the shadow process and the underlying database system(s);
- support for dynamic SQL;
- diagnostics management;
- transaction management;
- request processing in an environment of multiple heterogeneous database systems.

Trademarks

Acknowledgements

Many ideas in this paper were discussed at meetings of our research group. We would like to thank Dexter Bradshaw, Neil Coburn, Jan Pachl and Bob Sunday for their valuable suggestions and comments.

References

[1] ANSI-ISO, Database Language - SQL with Integrity Enhancement, ISO 9075-1989/ANSI X3.135-1989.

[2] ANSI-ISO, Database Language SQL2, ANSI X3H2, ISO/IEC JTC1 SC21 WG3 N985 (DBL SLC-1/SEL-2), December 1989.

[3] IBM, SQL/Data System: Application Programming for IBM VM Systems, Version 3 Release 2, 1991.

[4] ISO, Information Technology - Open Systems Interconnection - Remote Database Access - Part 1: Generic, ISO/IEC DP 9579-1, October 1990.

[5] ISO, Information Technology - Open Systems Interconnection - Remote Database Access - Part 2: SQL Specialization, ISO/IEC DP 9579-2, October 1990.

[6] D.P. Bradshaw et al., Transaction Management in Hermes using Camelot, Technical Report (in preparation), Department of Computer Science, University of Waterloo, 1991.

[7] P. Dadam et al., A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View of Flat Tables and Hierarchies, Proc. ACM-SIGMOD Conference, 1986, 356-367.

[8] P.-Å. Larson, The Data Model and Query Language of LauRel, Data Engineering, Quarterly Bulletin (September, 1988), Vol 11, No 3, 23-30.

[9] M.A. Roth, H.F. Korth, and D.S. Batory, SQL/NF: A Query Language for non-NF Relational Databases, Technical Report TR-85-19, Dept. of Computer Science, University of Texas at Austin, 1985.

[10] R.E. Strom, D.F. Bacon, A.P. Goldberg, A. Lowry, D.M. Yellin, and S.A. Yemini, Hermes: A Language for Distributed Computing, Prentice-Hall, 1991.

About the Authors

P.-Å. (Paul) Larson is a Professor in the Department of Computer Science, University of Waterloo, where he currently serves as chairman of the department. His research is focussed on database systems, in particular, file structures, query processing and optimization, and parallel and distributed databases. His Internet address is palarson@uwaterloo.ca.

Qiang Zhu is a Ph.D. student in the Department of Computer Science, University of Waterloo. He holds M.Eng. and M.Sc. degrees in Computer Science and Applied Mathematics, respectively. He was principal developer of a relational database management system. His current interests include distributed database systems and query optimization. His Internet address is qzhu@violet.uwaterloo.ca.

Frank Pellow is a senior development analyst at the IBM Laboratory in Toronto. He enjoys juggling his jobs as a member of IBM's SQL Language Council, a developer of the SQL/DS product, and a researcher in CAS. He enjoys construction projects on his island in northern Ontario even more. His Internet address is pellow@torolab3.vnet.ibm.com.

Appendix: Syntax of SQL Embedded in Hermes

Unless otherwise noted, this syntax is that of embedded SQL in ISO 9075-1989 (also called SQL89). Elements that are taken from SQL2 are flagged as such. The CORDS extensions appear in **bold face**. The syntactic notation is an extended version of BNF (Backus Naur Form) and is the same as the notation used in ISO 9075-1989.

source [1]	note [2]	line	syntax	cross reference
C	*a	0010	**<Hermes variable definition> ::= ... as defined in Hermes manual.**	
		0020	<embedded SQL declare section> ::=	
			<embedded SQL begin declare>	0030
C	*a		**<Hermes variable definition>...**	0010
			<embedded SQL end declare>	0060
		0030	<embedded SQL begin declare> ::=	
	*b		[<SQL prefix>] BEGIN DECLARE SECTION <SQL terminator>	0040 0050
		0040	<SQL prefix> ::= EXEC SQL	
		0050	<SQL terminator> ::= ;	
		0060	<embedded SQL end declare> ::=	
	*b		[<SQL prefix>] END DECLARE SECTION <SQL terminator>	0040 0050
		0070	<embedded SQL statement> ::=	
	*b		[<SQL prefix>]	0040
			{ <embedded exception declaration>	0080
			| <SQL statement> }	0120
			<SQL terminator>	0050
		0080	<embedded exception declaration> ::=	
			WHENEVER <condition> <exception action>	0090 0100
		0090	<condition> ::= SQLERROR	
			| NOT FOUND	
C			| **SQLWARNING**	
		0100	<exception action> ::= CONTINUE	
C			| **RAISE :<Hermes exception name>**	0110
C		0110	**<Hermes exception name> ::= Hermes identifier which identifies an exception name.**	
		0120	<SQL statement> ::=	
			<data statement>	0130
C	*c *d		| **<schema statement>**	0840
2	*e		| <transaction statement>	1160
2			| <connection statement>	1170
2	*f		| <dynamic statement>	1180
2	*g		| <diagnostics statement>	1190

[1] Blank means the source is SQL89; '2' means the source is SQL2; 'C' means the source is the CORDS project.

[2] Note references take the form of '*letter' to the left of the line to which they apply. All the notes may be found at the end of the list.

source	note	line	syntax	cross reference	
		0130	<data statement> ::=		
			<select statement: single row>	0140	
C			| **<select statement: all rows>**	0670	
			| <data change statement>	0740	
		0140	<select statement: single row> ::=		
			SELECT [ALL | DISTINCT] <select list>	0150	
			INTO { <select target list>	0380	
C	*m		| **RECORD :<embedded record variable name>** }	0390	
			<table expression>	0400	
		0150	<select list> ::= <value expression> [{,<value expression>}...]	0160	----
			| *		
		0160	<value expression> ::= <term>	0170	
			| <value expression> + <term>	----	----
			| <value expression> - <term>	----	----
		0170	<term> ::= <factor>	0180	
			| <term> * <factor>	----	----
			| <term> / <factor>	----	----
		0180	<factor> ::= + | - <primary>	0190	
		0190	<primary> ::= <value specification>	0200	
			| <column specification>	0250	
			| <set function specification>	0350	
			| (<value expression>)	0160	
		0200	<value specification> ::= <variable specification>	0210	
			| <literal>	0240	
			| USER		
		0210	<variable specification> ::= :<embedded unary variable name>	0220	
			[<indicator variable>]	0230	
		0220	<embedded unary variable name> ::= **Hermes identifier which identifies a variable of data type INTEGER, REAL or CHARSTRING.**		
		0230	<indicator variable> ::= [INDICATOR] :<embedded unary variable name>	0220	
		0240	<literal> ::= as defined in ISO 9075-1989		
	*k	0250	<column specification> ::=		
			[<qualifier>.] <column name>	0260	0320
C			| **:<embedded table variable name>.<embedded column name>**	0330	0340
		0260	<qualifier> ::= <table name> | <correlation name>	0270	0310
		0270	<table name> ::= [<authorization identifier>] . <table identifier>	0280	0300
		0280	<authorization identifier> ::= <identifier>	0290	
		0290	<identifier> ::= as defined in ISO 9075-1989		
		0300	<table identifier> ::= <identifier>	0290	
		0310	<correlation name> ::= <identifier>	0290	
		0320	<column name> ::= <identifier>	0290	

```
source  note | line  syntax                                                                  cross reference
C            | 0330  <embedded table variable name> ::= Hermes identifier which identifies
             |                                          a variable of a Hermes common table.
C            | 0340  <embedded column name> ::= Hermes identifier which identifies a
             |                                  component variable of a Hermes common table.
             | 0350  <set function specification> ::=   COUNT(*)
             |                                       | <distinct set function>          0360
             |                                       | <all set function>               0370
             | 0360  <distinct set function> ::=
             |           {AVG | MAX | MIN | SUM | COUNT}  ( DISTINCT <column specification> )  0250
             | 0370  <all set function> ::=
             |           {AVG | MAX | MIN | SUM}  ( [ ALL ]  <value expression> )          0160
             | 0380  <select target list> ::=
             |                 <variable specification>  [ {,<variable specification>}... ]  0210  ----
C            | 0390  <embedded record variable name> ::= Hermes identifier which identifies
             |                                           a variable of Hermes common record.
             | 0400  <table expression> ::=   <from clause>                               0410
             |                              [ <where clause> ]                            0430
             |                              [ <group by clause> ]                         0650
             |                              [ <having clause> ]                           0660
             | 0410  <from clause> ::= FROM <table reference>  [ {,<table reference>} ]   0420  ----
             | 0420  <table reference> ::= {  <table name>                                0270
C            |                             | :<embedded table variable name> }            0330
             |                            [ <correlation name> ]                         0310
             | 0430  <where clause> ::= WHERE <search condition>                          0440
             | 0440  <search condition> ::=   <boolean term>                              0450
             |                             | <search condition> OR <boolean term>         ----  ----
             | 0450  <boolean term> ::=   <boolean factor>                                 0460
             |                          | <boolean term> AND <boolean factor>           ----  ----
             | 0460  <boolean factor> ::= [ NOT ]  <boolean primary>                       0470
             | 0470  <boolean primary> ::= <predicate> | (<search condition>)              0480  0440
             | 0480  <predicate> ::=   <comparison predicate>                              0490
             |                       | <between predicate>                               0530
             |                       | <in predicate>                                    0540
             |                       | <like predicate>                                  0560
             |                       | <null predicate>                                  0590
             |                       | <quantified predicate>                            0600
             |                       | <exists predicate>                                0640
             | 0490  <comparison predicate> ::= <value expression> <comp op>              0160  0500
             |                                  { <value expression>  <subquery > }       ----  0510
             | 0500  <comp op> ::=  = | <> | < | > | <= | >=
             | 0510  <subquery> ::= ( SELECT  [ ALL | DISTINCT ]  <result specification>  0520
             |                       <table expression> )                                0400
             | 0520  <result specification> ::= <value expression> | *                   0160
```

source	note	line	syntax	cross reference
		0530	<between predicate> ::= <value expression> [NOT] BETWEEN	0160
			<value expression> AND <value expression>	---- ----
		0540	<in predicate> ::= <value expression> [NOT] IN	0160
			{ <subquery> | (<in value list>) }	0510 0550
		0550	<in value list> ::= <value specification>{,<value specification>}...	0200 ----
		0560	<like predicate> ::= <column specification> [NOT] LIKE	0250
			<pattern> [ESCAPE <escape charater>]	0570 0580
		0570	<pattern> ::= <value specification>	0200
		0580	<escape charater> ::= <value specification>	0200
		0590	<null predicate> ::= <column specification> IS [NOT] NULL	0250
		0600	<quantified predicate> ::= <value expression> <comp op>	0160 0500
			<quantifier> <subquery>	0610 0510
		0610	<quantifier> ::= <all> | <some>	0620 0630
		0620	<all> ::= ALL	
		0630	<some> ::= SOME | ANY	
		0640	<exists predicate> ::= EXISTS <subquery>	0510
		0650	<group by clause> ::=	
			GROUP BY <column specification> [{,<column specification>}...]	0250 ----
		0660	<having clause> ::= HAVING <search condition>	0440
C		0670	**<select statement: all rows> ::=**	
			SELECT [ALL | DISTINCT] <select list>	0150
C			**INTO TABLE :<embedded table variable name>**	0330
			<table expression> [UNION [ALL] <query expression>]	0400 0680
			[<order by clause>]	0710
		0680	<query expression> ::= <query term>	0690
			| <query expression> UNION [ALL] <query term>	---- ----
		0690	<query term> ::= <query specification> | (<query expression>)	0700 0680
		0700	<query specification> ::=	
			SELECT [ALL | DISTINCT] <select list> <table expression>	0150 0400
		0710	<order by clause> ::=	
			ORDER BY <sort specification> [{,<sort specification>}...]	0720 ----
		0720	<sort specification> ::=	
	*n		{ <unsigned integer>	0730
			| <column specification>} [ASC | DESC]	0250
		0730	<unsigned integer> ::= as defined in ISO 9075-1989	

```
source  note  line  syntax                                                                  cross reference

              0740  <data change statement> ::=
                          <insert statement>                                              0750
                        | <delete statement: searched>                                    0800
                        | <update statement: searched>                                    0810

              0750  <insert statement> ::= INSERT INTO <insert object>                    0760
                                            { VALUES (<insert value list>)              0780
C       *m                                  | RECORD :<embedded record variable name>   0390
C                                           | TABLE :<embedded table variable name>     0330
                                            | <query specification>       }             0700

              0760  <insert object> ::= {  <table name>  (<insert column list>)          0270  0770
                                         | :<embedded table variable name> }           0330

              0770  <insert column list> ::= <column name>  [ {,<column name>}... ]     0320  ----

              0780  <insert value list> ::= <insert value>  [ {,<insert value>}... ]    0790  ----

              0790  <insert value> ::= <value specification> | NULL                     0200

              0800  <delete statement: searched> ::=
                          DELETE FROM {  <table name>                                   0270
C                                     | :<embedded table variable name> }             0330
                            [ WHERE <search condition> ]                               0440

              0810  <update statement: searched> ::=
                          UPDATE {  <table name>                                        0270
C                                 | :<embedded table variable name> }                 0330
                            SET <set clause> [ {,<set clause>}... ]                    0820  ----
                            [ WHERE <search condition> ]                                0440

              0820  <set clause> ::= <object column> = {<value expression> | NULL}      0830  0160

              0830  <object column> ::=   <column name>                                 0320
C                                       | :<embedded column name>                     0340


      *c *d   0840  <schema statement> ::=
      *h              <create table statement>                                          0850
2                   | <drop table statement>                                           1000
2                   | <alter table statement>                                          1010
      *i            | <create view statement>                                          1060
2                   | <drop view statement>                                            1080
      *j            | <grant statement>                                                1090
2                   | <revoke statement>                                               1150

      *h      0850  <create table statement> ::=
                      CREATE TABLE <table name>                                        0270
                        ( <table element>  [ {,<table element>}... ] )                 0860  ----

              0860  <table element> ::=   <column definition>                          0870
                                         | <table constraint definition>               0950

              0870  <column definition> ::= <column name> <data type>                   0320  0880
                                             [ <default clause> ]                      0890
                                             [ <column constraint>... ]                0900

              0880  <data type> ::= as defined in ISO 9075-1989.

              0890  <default clause> ::= DEFAULT { <literal> | USER | NULL }           0240
```

source	note	line	syntax	cross reference
		0900	<column constraint> ::= NOT NULL [<unique specification>] | <references specifiaction> | CHECK (<search condition>)	0910 0920 0440
		0910	<unique specification> ::= UNIQUE | PRIMARY KEY	
		0920	<references specification> ::= REFERENCES <referenced table and column>	0930
		0930	<referenced table and column> ::= <table name> [(<reference column list>)]	0270 0940
		0940	<reference column list> ::= <column name> [{,<column name>}...]	0320 ----
		0950	<table constraint definition> ::= <unique constraint defintion> | <referential constraint definition> | <check constraint definition>	0960 0980 0990
		0960	<unique constraint definition> ::= <unique specification> [(<unique column list>)]	0910 0970
		0970	<unique column list> ::= <column name> [{,<column name>}...]	0320 ----
		0980	<referential constraint definition> ::= FOREIGN KEY (<reference column list>) <references specification>	0940 0920
		0990	<check constraint definition> ::= CHECK (<search condition>)	0440
2		1000	<drop table statement> ::= DROP TABLE <table name> [CASCADE]	0270
2		1010	<alter table statement> ::= ALTER TABLE <table name> <alter action>	0270 1010
2		1010	<alter action> ::= <add column defintion> | <drop column defintion> | <replace default clause>	1030 1040 1050
2		1030	<add column definition> ::= ADD <column definition>	0870
2		1040	<drop column definition> ::= DROP <column name> [CASCADE]	0320
2		1050	<replace default clause> ::= REPLACE <column name> <default clause>	0320 0890

	*i	1060	<create view statement> ::=
			  CREATE VIEW <table name> [(<view column list>)]
			  AS <query specification>
			  [WITH CHECK OPTION]
				0270 1070
				0700

		1070	<view column list> ::= <column name> [{,<column name>}...]	0320 ----
		1080	<drop view statement> ::= DROP VIEW <table name> [CASCADE]	0270

```
source  note | line  syntax                                                                  cross reference
        *j   | 1090  <grant statement> ::=
                       GRANT <privileges>  ON <table name>                                   1110  0270
                         TO <grantee>  [ {,<grantee>}... ]  [ WITH GRANT OPTION ]            1140  ----

             | 1110  <privileges> ::=   ALL PRIVILEGES
                                      | <action>  [ {,<action>}... ]                        1120

             | 1120  <action> ::=   SELECT | INSERT | DELETE
                                   | UPDATE  [ (<grant column list>) ]                      1130
                                   | REFERENCES  [ (<grant column list>) ]                  ----

             | 1130  <grant column list> ::= <column name>  [ {,<column name>}... ]        0320  ----

             | 1140  <grantee> ::= PUBLIC | <authorization identifier>                     0280

2            | 1150  <revoke statement> ::=
                       REVOKE  [ GRANT OPTION FOR ]  <privileges>  ON <table name>          1110  0270
                         FROM <grantee>  [ {,<grantee>}...]  [ CASCADE ]                   1140  ----

2       *e   | 1160  <transaction statement> ::=    ... not determined yet

2            | 1170  <connection statement> ::=   ... not determined yet

2       *f   | 1180  <dynamic statement> ::=    ... not determined yet

2       *g   | 1190  <diagnostics statement> ::=   ... not determined yet
```

Notes:

*a In the SQL/Hermes embedding, host variables may be declared anywhere, that is the Declaration Section is optional. In the standards, host variables must be declared in a Declaration Section.

*b In the SQL/Hermes embedding, the <SQL prefix> is optional. It is not optional in the standards.

*c <schema statement>s are called <schema definition>s in ISO 9075-1989 and in SQL2 and they are not suppor in embedded SQL.

*d <embedded table variable name>s and <embedded column name>s) can not be used in schema statements.

*e SQL2 contains <transaction statement>s. However, transaction management in CORDS is intended to be wide in scope than SQL2, so our <transaction statement>s may vary.

*f We have not yet determined the method that CORDS will use to support dynamic creation and execution of SQL statements.

*g We have not yet determined whether or not CORDS will utilize some form of the SQL2 <diagnostic statement>s.

*h <create table statement>s are called <table definition>s in ISO 9075-1989 and in SQL2.

*i <create view statement>s are called <view definition>s in ISO 9075-1989 and in SQL2.

*j <grant statement>s are called <privilege definition>s in ISO 9075-1989 but <grant statement>s in SQL2.

*k Some syntactic elements support a list of <column specification> elements (for instance, see the <group by> clause on line 980). Only one of these forms of <column specification> is allowed per list.

*m Selecting into a record provides the same capability as host structure support in products such as DB2.

*n The use of an integer to identify the ordinal position of the column being sorted is a depricated feature in SQL2. The need for it is removed by support of the AS clause when defining a derived column. We may choose to support this instead in the SQL/Hermes embedding.