REAM: An SQL Based and Extensible Relational Database Management System

Suguru KAWAKAMI, Takashi NAKAYAMA, Koichi KASHIWABARA and Sadayuki HIKITA

OKI Electric Industry Co., Ltd.
Warabi, Saitama 335, JAPAN

Abstract

REAM (**RE**lational d**A**tabase **M**anagement system) is a relational database management system developed by OKI. The relational database has been expected to be widespread thanks to its well-defined mathematical foundation; however, application programmers have suffered from incompatible access interfaces among various so-called relational database systems so far due to the absence of the international standard. The access interface of REAM is an implementation of the internationally standardized SQL. REAM also is extensible in the sense that it may be accessible through other access languages besides SQL. In addition, REAM can be extended to distributed environment. For the sake of extensibility, REAM comes up with a two layered interface structure, the SQL language interface and the REAM kernel interface. By adding another language interface on the top of the REAM kernel interface, REAM easily provides another language interface for application programs. REAM has already had a set of extended functional capability so that it can work as an underlying system for a distributed database system.

This paper presents the system's design decisions and extended functional capability with addressing the problems left in the standardized SQL. Then, the current implementation of REAM is explained.

1. Introduction

We have developed a relational database management system called REAM on OKITAC 8300, a mini computer produced by OKI. REAM features the standardized SQL language based access interface. SQL, a relational database access language, became an international standard in 1987 by the International Organization for Standardization (ISO) [ISO87]. It has also become a Japan Industrial Standard (JIS) in 1987. In remainder of this paper, we call the standardized SQL as the standard SQL. This standardization brought us a great benefit in development of database applications. In prior to the standard SQL, we have already had db-1, a relational database machine, *FREND*, the predecessor of db-1 and DBMS-E [Yamada81], a database management system on OKITAC-50V. Due to their incompatible access interface, all of them have forced us to afford reimplementation efforts of application programs. The standard SQL is supposed to greatly reduce this overhead.

However, it has turned out that a pure implementation of the standard is not sufficient as an access interface of a relation database system. REAM is an extensible database management system. By an extensible database management system, we mean REAM may support other access language than the standard SQL and can be extended to a distributed database system. In this paper, we present the goals of REAM and the software structure we adopted in REAM to accomplish those goals.

2. Goals

Our primary goal is to provide the user with an access interface based on the standard SQL syntax. However, the access interface of REAM is an extension of the standard, not a pure implementation. We assumed the major applications of REAM were, first, transaction based programs, typically an application called Electric Data Processing (EDP), and interactive processing, such as a decision supporting system. REAM is designed to deal with both types of applications with good performance. Unfortunately, the standard SQL is not sufficient enough to perform processing both of them. A transaction based application requires high performance and high degree of concurrency control simultaneously. It usually manipulates a rather small number of records. In contrast, an interactive application does not have a deterministic accessing pattern. A large number of records are accessed on processing queries from the user. Performance does not matter in this type of applications. Subsequently, we realized that our implementation should complement the standard SQL so as to fulfill those, sometimes contradicting, requirements.

Another important goal is to support multi access languages in addition to the standard SQL. We do not think the current standard is the eventual goal of the access language for a relational database. As C. J. Codd pointed out [Date84], SQL has several problems in accessing a relational database. In fact, SQL2, a super set of the current standard, has already been on the way to another international standard. Forth generation languages are other candidates for the access language. A standard SQL oriented implementation was not supposed to be a good choice in practical implementation. We, thus, intended to implement a set of extended access primitives so that the standard SQL based interface could be implemented on the top of those access primitives.

Distributed database system is coming of age. Extensibility to distributed environment is an important issue in REAM. A database, once enshrined in a host computer, is distributing among cheap but high performance workstations

which are networking each other. A distributed database is expected to be in practical use in next few years.

3. Software Structure

Figure-1 depicts the two layered software structure of REAM. We adopted this structure for the sake of extensibility. The upper layer, the SQL language interface, is responsible for providing the application program with an SQL based access interface. The lower layer, the REAM kernel interface, supports the advanced primitives of the REAM kernel which performs database management. In this section, we describe the functionality of each interface layer and how the goals we set are accomplished by this structure.

3.1 SQL Language Interface

The standard SQL is expected to be extended in near future. For the sake of conformance of the current standard, the SQL interface is separated into two software modules. The standard SQL interface is carefully designed so that the conformance is guaranteed and the future extension of the current standard will not result in a major change of the entire REAM. The extended SQL interface will absorb the expected extension of the access interface. Japanese data type, for example, is support in this interface.

3.1.1 Standard SQL Interface

This interface provides the Data Manipulation Language. Although the standard defines both the Data Manipulation Language (DML) and the Data Definition Language (DDL), we decided it was inefficient for a commercial implementation to have both of them in a same language interface. The DDL appeared not to accomplish satisfiable performance due to the lack of crucial features, for instance, capability of indexing. Besides, the DDL specifies a schema definition and manipulation. A schema is merely accessed or changed by an application program. Implementing the DDL and the DML separately does not hurt the portability of applications.

3.1.2 Extended SQL Language Interface

We have extended the SQL based interface so that REAM can efficiently support both of transaction based processing and interactive processing. The standard, however, lacks several vital aspects in supporting the assumed applications. We considered we should extend the standard in the following three aspects, say, concurrency control, dynamic SQL and Japanese data type support, so long as those extension do not hurt the conformance of the standard interface.

Table-1: Implementation Method for Isolation Levels

Level	Implementation Method
1	No Lock
2	Not Implemented
3	Not Implemented
4	Lock by Tuple
5	Lock by Relation

Concurrency Control

First, the standard SQL does not effectively guarantees high degree of concurrency control. As stated, high degree of concurrency control is indispensable for transaction based processing to provide better performance. Concurrency control is still being discussed in standardization. Implementing it in advance may lead to an incompatible implementation with the future standard. However, it is worthwhile supporting high degree of concurrency control in a commercial based database system. Besides, there show up only one or two concurrency control related statements in an application. It is supposed to be easy to modify them along with a new standard in case of that REAM's concurrency control is incompatible.

REAM features the multi-leveled concurrency control scheme so that an application can make the best choice of the degree of concurrency control. A database management system sometimes has to be faced a serious dilemma that it must simultaneously perform high degree of concurrency control and guarantee consistent data. This dilemma is never solved as long as the database management system depends upon a single level concurrency. A transaction based application requires higher concurrency with placing high performance being the first rated requirement. It is tough to fulfill both requirements with guaranteeing data consistency.

REAM supports the three leveled concurrency control shown in Table-1. Those three levels are respectively corresponding to the level1, 4 and 5 in the five isolation leveled control which has been introduced by J. N. Gray [Gray76]. In the Gray's report [Date83], it is said that the level 5 is inefficient. It also reported that the level 2 and 3 can be effectively covered with the level 4 concurrency.

Dynamic SQL

Run-time interpretation of SQL statements is available in REAM. The standard lacks dynamic SQL primitives. In the current standard, SQL statements are supposed to be embedded in a host language program. The embedded SQL only allows the user to change values in search conditions in run-time. SQL statements, rather than values, should be able to be changed in run-time in an interactive application, in which access is done by try-and-error basis. Run-time interpretation is necessary to deal with an interactive application. In addition, the Remote Database Access Protocol (RDA) would facilitate a basic standard interworking of distributed database systems [ISO88]. Its protocol basically consists of a sequence of SQL statements, which must be dynamically interpreted.

Japanese Data Type Support

Furthermore, only alphanumeric data type is considered in the current standard. REAM, on the other hand, introduces Japanese data type. For non-alphanumeric-natives, the lack of capability of handling national languages is a serious drawback. By non-alphanumeric-natives, we assume people whose mother tongue is represented with multi-byte code systems. For example, a two-byte code system represents the Japanese characters set. In the sense that the character type is supposed to be a one byte code, the standard does not effectively support national languages. This drawback may prevent the standard from widespreading in non-alphanumeric-native countries.

Actually, we had three choices in extending the SQL for Japanese language support. The choices and the reasons of our choice are summerized below. Our choice was (1) as the first implementation.

(1) Data Type Support

The Japanese data type may be stored and accessed in a database. This extension makes minimum Japanese processing available, even though the readability and flexibility are limited.

(2) Constant Support

Japanese constant is allowed in the SQL syntax. The standard allows the user to write constants in a search condition. This choice makes the SQL more powerful in describing database operations. Nonetheless, conformance and portability might be victimized by this extension, because national language constants may be allowed in SQL2 [ISO88]. We were afraid that Japanese constant support in prior to SQL2's standardization might lead to an unconformable implementation.

(3) Identifier Support

An identifier can be written in Japanese. The standard only allows the user to write either a relation name, an authorization identifier, a corelation name or column names in alphanumeric code. Japanese-written identifier improves the readability of a database application; however, as addressed above, conformance and portability may get hurt by allowing Japanese identifiers in an application.

3.2 REAM Kernel Interface

This interface has extended functionalities in performing database management. The REAM kernel interface is a collection of function calls which provide basic database management primitives. One of the objectives of the REAM kernel interface is distributed toward extensibility. Another one important objective is architecture independence. Architecture independence means that the REAM kernel interface is available in accessing not only a relation but files of other types, for example, a sequential file.

The primitives are categorized into two groups, namely, the data access primitives and the transaction control primitives. The data access primitives perform file access. The accessed file is either a relation or non-relation type file. It is unnecessary for the user to distinguish the difference between the physical file structure which he/she is accessing. The transaction control primitives are indispensable for a database management system to effectively run an application to be extended to distributed environment. The concept of transaction and session is supported.

3.2.1 Data Access Primitives

The data access primitives are more than the standard SQL statements. Performance requirement necessitates more primitives than those defined in the standard. Three major extension has been done to this kernel primitive criteria. One is the cursor specified tuple insertion. One is the temporary relation capability. Another one is the tuple identifier support.

Cursor Specified Tuple Insertion

This primitive is capable of inserting a new tuple to a relation with a cursor specification. In the standard SQL, a cursor is supposed to always point an existing tuple, which implies the semantics cannot tolerate a cursor specified tuple insertion. Nonetheless, this restriction considerably reduces the performance in inserting operation, especially in case of insertion of huge amount of new tuples.

Temporary Relation

A temporary relation can be arbitrarily created in a database session and destroyed as the session is terminated. A temporary relation of REAM takes advantage over the ordinal relation of the standard. First, it is unnecessary for a temporary relation to get logged, which implies that REAM can operate a temporary relation more faster than to handle an ordinal relation. Moreover, a temporary relation name needs to be unique only in a session, not in a database. An interactive application generates a bunch of relations in prior to reaching the final result. It is preferable that those scratch pad relations disappear when the application finishes processing queries without producing unnecessary overhead. Because a temporary relation is automatically destroyed when a session is terminated, it is suitable to store those scratch pad relations.

Tuple Identifier

The REAM kernel manages a tuple with an identifier (tuple ID) in cases for performance reason in distributed environment. A tuple ID specifies a unique tuple in a database. Suppose that an application is located on a remote site from a database site and tries to access the database through the REAM interface. It is obvious that accessing a remote relation on tuple by tuple basis produces a considerable communication overhead on exchanging tuples back and forth between the application site and the database site. The well-known fact shows us we would better read the remote data as much as possible at a time [Hikita85b]. Unfortunately, in the standard SQL semantics, the cursor has no choice but has to move from the previous spot, if the application reads more than a tuple at a time. This nature makes it very hard to find the previous spot the cursor used to point after the application finishes modifying all of tuples read from the remote site. By using a tuple ID, the tuple to be updated can easily found on the database site.

With the extended concurrency control mechanism we ever introduced [Hikita84], the tuple ID would accomplish higher degree of concurrency control in distributed environment. The target relation can be cached on the application site and is modified, meanwhile other transactions are allowed to access to the original copy on the database site. After finishing the transaction on the application site, the modified tuples are sent back to the database site with their tuple ID's. The relation is updated along with referring the tuple ID's. Of course, a carefully designed control must guarantee the consistent state. The communication overhead is greatly reduced, and the degree of concurrency is remarkably improved by means of active use of the tuple ID.

3.2.2 Transaction Control Primitives

The transaction control is another functional capability the current standard SQL lacks, even though the concept of transaction is mentioned. The transaction control primitives are prepared for supporting not only the centralized transaction but also the decentralized ones.

Session Control

A session in the REAM kernel begins when the user starts accessing the kernel and ends as the user issues the session termination primitive. A start of a session, however, is not explicitly declared in the standard. Without explicit session control primitives, it is hard to control the underlying communication software.

Two Phase Commit

The capability of two phase commit makes REAM be an underlying system for a to-be expected distributed database system. Two phase commit is supported by the combination of the secure, the commit and the abort primitives. The capability of each primitives is to be compatible with the correspondents defined in another standard discussed by ISO [ISO87b].

4. Structure of REAM

REAM consists of three software modules illustrated in Figure-1. The SQL preprocessor provides the standard SQL based access interface. The preprocessor is in charge of translating embedded SQL statements into a program in a host language. The SQL access processor issues the REAM kernel calls, and the REAM kernel performs database management.

4.1 SQL Preprocessor

The SQL preprocessor is an implementation of the standard SQL. As the first implementation, the level 1 DML has been supported, which, of course, is not our eventual goal. Currently, COBOL and C are supported as host languages. The standard SQL does not define the C embedded SQL, but, considering the widespread use of C, we decided to provide the C embedded SQL based on the SQL2 specification [ISO88b].

4.2 SQL Access Processor

The SQL access processor performs the extended SQL functions. It is called from an application to translate function calls into a set of corresponding REAM kernel primitives. The access processor, then, receives tuples from the REAM kernel to sort, to perform grouping and to make sub-queries based upon them.

4.3 REAM Kernel

Our efforts to improve the performance has been mainly concentrated on the cache management and the log management in the REAM kernel. the performance of the kernel dominates the system performance of the entire REAM. It is clear that the number of secondary storage access is the dominant factor in database system performance. Both the cache management and the log management are tightly related to disk accessing. Careless design often produces frequent disk access and results in poor performance.

4.3.1 Cache Management

The REAM kernel has its own pager of the database cache independent from the underlying operating system. In a database application, the locality of accessing is not always maintained. The locality of accessing is, on the other hand, the rationale of the replacement algorithm the underlying operating system adopts. Rather, a relation is sometimes accessed completely at random. Without being aware of the application's accessing behavior, it is almost impossible for an operating system to perform an optimal caching. Instead, the REAM kernel replaces the contents of the cache based on the hueristic knowledge about the typical accessing pattern of database applications. For instance, the REAM kernel pager allocates more caching space to indexes than relations. An index has better locality, meanwhile a relation often does not.

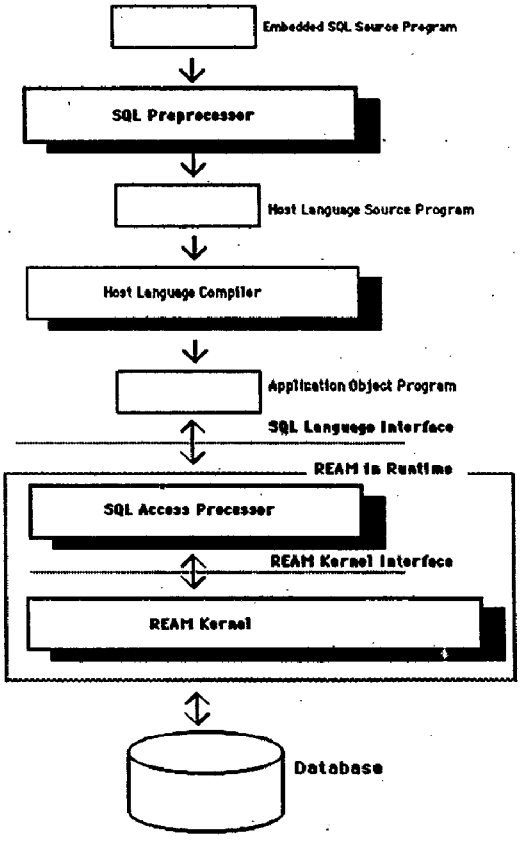

Figure- 1: The Two Layered Interface and Software Structure of REAM

4.3.2 Log Management

The REAM kernel supports two kinds of recovery scheme, a sort of shadowing [Lorie77] and logging. The recovery scheme is dynamically selected by a transaction depending upon a concurrency control level. At the level 4 concurrency, logging is effective, because the transaction is expected to modify a small number of tuples. It is preferable that all tuples in a same relation are located in a continuous space in secondary storage for quick accessing. Should the REAM kernel have performed shadowing tuple by tuple, the modified tuples would have been scattered somewhere else from other unmodified tuples. In contrast, the transaction with requires the level 5 concurrency is expected to update a large part of a relation. In this case, an entire relation should be shadowed and reallocated to another continuous secondary storage space after the transaction is committed.

5. Future Work

First of all, the immediate extension is to enhance REAM to support the level 2 DML of the standard SQL. REAM also should include SQL2 along with the standardization of it. Some SQL2 toward primitives have been already prepared in the REAM kernel to absorb the features to be standardized in SQL2, even though some aspects, for example, national language support, is still unclear. In next few years, we plan to integrate REAM with a distributed database management system. A set of primitives for this integration has been implemented in the REAM kernel. Besides them, REAM will be enhanced with the extended concurrency control [Hikita84]. We now intend to justify our design decisions in distributed environment.

6. Summary

In this paper, we presented the rationales of our design decisions in implementing REAM, a relational database management system based on the international standard SQL. It is thought that the standard SQL is solving the incompatible access interface woe in relational database applications. On making design decisions, we realized that the current standard could not be the eventual goal. It turns out that the current standard lacks several crucial functionalities for a commercial product. That recognition motivated us to improve REAM to an extensible database management system with supporting the standard SQL interface. We also described the software structure we adopted to support our decisions. REAM is now on the way to the distributed database world.

Acknowledgement

The authors would like to thank Kuniko Saito, Akifumi Sakamoto and Yuichiro Hiranuma. The discussion with them let us recognize the importance of distributed toward extensibility in a future database system. Satoshi Shimizu and Yoshifumi Yamada have greatly contributed to the implementation of the SQL access processor. We also are indebted to Makoto Kotera for valuable comments on this paper.

References

[ISO87] International Organization for Standardization, *International Standard: Information Processing System - Database Language SQL*, ISO9075, June 1987.

[Hikita85] Hikita, S. Kawakami, S. and Haniuda, H., Database Machine FREND, In *proceedings of 4th International Workshop on Database Machines.*, p.p.190-207, 1985.

[Hikita87] Hikita, S. Kawakami, S., Sakamoto, A. and Matsushita, Y., An Approach for Customizing Services of Database Machines., In *proceedings of 5th International Workshop on Database Machines.*, p.p.305-318, 1987.

[Yamada81] Yamada, R., Tateoka, K., Mori, S. and Nakae, Y., Development of a Relational-like Database Management System for Mini-/MICRO Computers., In *proceedings of COMPSAC81.*, p.p.403-410, October 1981.

[Date84] Date, C. J., A Critique of the SQL Database Language., *SIGMOD RECORD*, Vol.14., No.3, p.p.8-54, 1984.

[Gray76] Gray, J. N., Lorie, R. A., Putzolou, G. R. and Traiger, I. L., Granularity of Locks and Degree of Consistency in a Shared Data Base, In *proceedings of IFIP TC-2 Working Conference on Modeling in Database Management Systems.*, January 1976.

[Date83] Date, C. J., *An Introduction to Database Systems Volume II.*, ADDISON WESLEY, p.p.83-142, 1983.

[ISO88] International Organization for Standardization, *ISO/IEC JTC/SC21 Information Retrieval, Transfer Management for OSI.*, July 1988.

[Hikita85b] Hikita, S. Kawakami, S. and Haniuda, H., A Stepwise Approach to Distributed Database Systems by Database Machines., In *proceedings of 1985 ACM Sigsmall Symposium on Small Systems.*, p.p.18-24, May 1985.

[Hikita84] Hikita, S. Kawakami, S. and Sano, K., Extended Functions of the Database Machine FREND for Interactive Systems., In *proceedings of 1984 IEEE Computer Society Workshop on Visual Languages.*, December 1984.

[ISO87b] International Organization for Standardization, *ISO/DIS9805/2: Information Processing Systems - Open Systems Interconnection - Protocol Specifications for Common Application Service Elements - Commitment, Concurrency and Recovery.*, December 1987.

[ISO88b] International Organization for Standardization, *ISO-ANSI (working draft) Database Language SQL2.*, July 1988.

[Lorie77] Lorie, R. A., Physical Integrity in a Large Segmented Database, *ACM Transaction on Database Systems.*, Vol.2, No.1, p.p.91-104, 1977.