CONCEPT FOR A REAL-TIME STRUCTURED DATABASE QUERY LANGUAGE (RT-SQL)

PAUL J. FORTIER* **and JANET PRICHARD†**

**Naval Undersea Warfare Center, Newport, Rhode Island, 02841-1708, USA*
† The University of Rhode Island, Computer Science Department, Kingston, R.I. 02881, USA

Abstract.

Real-time database management systems have become a hot topic in the research and development community of late (Fort94a,Grah93,WCPP93). In addition there has been a movement in the standards community to examine and develop extensions to existing and proposed query languages to support real-time (Fish94,Fort94,Fort94a,FS94,Gord94).

This paper examines the state of research into real-time database management systems in the areas of database structuring, transaction structuring, transaction processing, concurrency control, recovery and real-time transaction scheduling. We then extend the findings and trends of this work into the high level specification of data definition language, data manipulation language and data control language extensions for the standard SQL2 and emerging SQL3 database query languages.

Keywords. Real-time, Database, Query language, SQL2, SQL3

1. INTRODUCTION

A real-time computing system differs from non-real-time systems in fundamental ways. A real-time system is required to interact with a physical system through extraction of sensor induced information, computation of some control, status or feedback information and to effect predictable reactive responses on the physical system within specified time frames using computed data. Real-time computing systems are being applied to a wide range of physical systems, such as automotive control, aircraft control, power management, automated factories, medical assistance and defense oriented systems. A database within a real-time computing system has the same requirements levied on it. A real-time database management system must continue to provide consistency and correctness of data manipulations, but must additionally be predictable, meet application computational correctness conditions and be timely with responses to application queries.

Traditional database management solutions are incompatible with this set of operational requirements. We cannot expect predictable, correct nor timely database query response if serializability, blocking, aborts and backward recovery are applied to transaction executions over a monolithic database. Therefore real-time database management requires an altered model of database structuring, transaction structuring, transaction execution, transaction recovery and consistency and correctness criteria that utilizes real-time applications operational requirements (embedded semantic information) to redefine these terms. Database management policies and mechanisms must be driven by the needs of the applications not by the needs of the database management system.

For real-time to become part of a database management systems requires altering the specification of standard database query languages to include real-time database management features for timeliness, correctness, consistency, predictability and recovery.

The focus of the remainder of this paper is to define the requirements of real-time database management and to outline how these requirements can be included in an altered SQL specification. The discussion will be high level and assumes a familiarity with the SQL2 (DD92) and the evolving SQL3 (MS92) standards.

2. REAL-TIME DATABASE MANAGEMENT

In this section we review pertinent research and developments in real-time database management and standardization efforts. We focus our discussion on areas that potentially have the greatest impact on database query language structure.

2.1 Real-time Scheduling

An important aspect of real-time computing is real-time scheduling. A real-time scheduler orders a set of ready to execute tasks and selects one for immediate service. A real-time scheduler is composed of a scheduling policy, describing what parameters to optimize selection on, how to handle overload and contention situations, and a mechanism implementing the policies selected. Research into real-time schedulers have taken two paths:

1. Static schedule generation
2. Dynamic schedule generation

In static scheduling the set of tasks is known apriori, and an optimal schedule for the set of tasks can be determined based on the policies being applied. The static schedule cannot handle ad hoc conditions nor error conditions that may arise. Therefore this type of scheduling policy is not being considered for this paper.

We focus instead on dynamic schedulers which allow for preemption and alterations of existing schedules based on dynamically changing system loadings and requirements. Numerous researchers (FA94,HS93,SS93) have examined a variety of parameters and techniques for dynamic real-time task scheduling policies and mechanisms.

Database research using real-time scheduling looks at the issues involved in and solutions for integrating real-time task schedulers with transaction management
(concurrency control) (AG88,Fort94a,Naka93,Son88) There are two elements to real-time transaction scheduling. The first deals with how to determine which transaction, from a set of waiting transactions to begin execution. The second element is to determine what transaction operation to execute next (concurrency control operations ordering) based on real-time, predictable and fault tolerant policies in place. Research indicates the need to alter the conventional model used for transaction and concurrency control scheduling to support the unique need of real-time database applications. We focus in this paper on policies to select transactions and transaction operations for execution based on real-time applications needs.

¿From the RT-SQL perspective, these studies indicate the need for language extensions to schedule transactions and transaction operations based on the notion of time, data dependencies, various database constraints and transaction structuring and processing constraints within the database.

2.2 Database Structuring

Real-time researchers have examined the need for finer grained partitioning of the database and reduced coupling of database partitions to provide increased availability of data with less blocking of data (Fort93,Herl90,KLS90,WCPP93) The main emphasis in these studies is the increased performance benefit realized by transaction concurrency and data availability through a finer grained and loosely coupled database structures.

A real-time SQL will be required to support the definition and use of finer grained loosely coupled database structures. This implies an alteration to the data definition language and data manipulation language to support decomposition within the specification and runtime management of the database and its data items.

2.3 Transaction Structuring

Transactions represent database units of work viewed by users as possessing the *ACID* properties (BHG87,OV91). The *ACID* properties include: Atomic, Consistent, Isolation and Durability of transaction execution. Current research for real-time focuses on altering these properties for the support of applications real-time and predictable needs. Major research areas include: transaction decomposition, transaction consistency redefinition and alteration of transaction correctness criteria using applications semantic database access needs (Fort93,KLS90,WCPP93).

Current research points out the need to alter the transaction specification model in SQL to provide for nested or partitioned transactions, inclusion of transaction related consistency constraints predicates and definitions for transaction correct execution that may not include strict serializability nor fully consistent database states. For predictable execution the SQL specification requires the ability to specify execution time, memory placement, secondary storage access and other resource limitation requirements within its boundaries for use by real-time off-line optimization packages.

2.4 Concurrency Control

Concurrency control within databases is used to ensure database consistency and correctness while allowing a set of transactions to execute concurrently. Concurrency control performs the task of determining how to interleave a set of database operations from multiple transactions based on conflict resolution policies and transaction correctness criteria for the set of database operations (BHG87,Fort93). Conventional concurrency control dealt only with transaction syntactic information and the guarantee of correctness and consistency by serializability of concurrently executing transactions. The problem with conventional concurrency control lies in their blind application of serializability to all transactions regardless of a transaction's semantic intentions.

Research has pointed out the flaw in this theory for real-time systems. Real-time database management requires concurrency control protocols that allow for

increased concurrency and early commit of partitioned transactions based on the real-time and predictable requirements of the applications running within the real-time system.

This requirement will cause the need to alter how an execution model for transaction processing is selected and constructed. The impact on SQL will be in the specification of what concurrency control protocol to employ for a database or partition thereof, to specify conditions upon which a transactions subparts can be specified, conditions for transaction predictable executions, what scheduling policy to apply and conditions when early commit can be allowed.

2.5 Transaction Recovery

On a system failure primary memory is lost and transactions are left in one of two states: committed before the failure, and active but not committed before the failure. Recovery must ensure committed transactions remain committed, database consistency and correctness are not violated upon a failure and restarted active transactions do not cause loss of database consistency. Failed transactions should not cause other transactions to read inconsistent data or cause other transactions to fail.

Conventional means for recovery of committed and active transactions use check pointing of data items, with redo for committed and undo for active transactions. Redo re executes the database actions of a committed transaction to make its changes permanent. Undo rolls back any changes to the database to restore a state that existed before the transaction executed. This model of recovery is not adequate for real-time database management systems where availability and timeliness of data may be more important than strict serializability. Correctness criteria for transactions execution and recovery must be altered to support the unique needs of real-time databases (Fort93,KLS90,SYKI92). It may be more desirable in the real-time database environment to do nothing, do an application provided recovery, or recover to a future correct state (LL88) using forward recovery techniques.

The state that existed at the time of transaction execution may not be the *correct* database state now, based on real-world alterations of the database. Database recovery should only affect the failed or erroneous portion of the database and transaction not the entire database and entire transaction load.

Transaction recovery policies and mechanisms for real-time need to be added to the specification of SQL's data (data temporal consistency constraints and enforcement rules) definition language and to the specification of SQL's data manipulation language for transaction (transaction temporal consistency constraint and enforcement rules) specification. The ability of transaction writers to specify exception conditions for software, transaction aborts, conflict resolution, and hardware faults must be added to the language also.

2.6 Database Standardization

Standardization efforts for database management systems is relatively new. The first major database management systems standards to emerge were the database language NDL for network model database applications and the database language SQL for relational model database applications in 1986. These initial standards provide a data definition language (DDL) and a data manipulation language (DML) for creating, populating and accessing data stored within the structured network or relational database. Since these initial standards, efforts have been underway to refine these standards (SQL2) and to fit these standards into new environments (SQL3) (Gall91,Gall92).

Beyond these active standard improvement programs, there are numerous sanctioned standards study efforts. These study programs have as their charters to define where these and other standards need to change to support emerging database management systems needs. Two related real-time database standards study activities, the ANSI database systems study group (DBSSG) predictable real-time interface systems task group (PRISTG) and the U.S. Navy's next generation computer resources (NGCR) database management interface standards working group (DISWG) fall into this category (PS94,Fort94b). These standards bodies are developing recommendations for real-time and next generation database management systems standards.

3 EXTENTIONS FOR REAL-TIME SQL

We focus this section of the paper on our ideas for alterations to the SQL2 and SQL3 database definition, control and manipulation languages to support real-time and predictability. The discussion will be broken up into changes to the data definition language, the data manipulation language and the data control language portions of the SQL specifications.

3.1 Data Definition Language Needs

Real-time systems have been constructed for a variety of applications areas and programming languages. Applications such as weather forecasting, medical monitoring, defense systems use a variety of non standard data types specific to their applications domain. to support these specialized data types and abstract data types supported by modern programming languages, the SQL data types must be extended.

To provide for non-traditional data types (BLOBS) partitioned database specification, data consistency, referential consistency, temporal consistency, fault tolerance, data distribution, heterogeneous databases and real-time service SQL's data definition language

and resulting data structures and controlling procedures need additional features and definitional flexibility. In the following subsections these constructs are examined separately.

3.1.1 ADT and BLOBs

An extension to SQL needed by most modern real-time programming languages and applications developers are abstract data types (ADT) and binary large objects (BLOBs). These data types are stored in the database, but interpreted by applications. The database does not keep copious amounts of meta data on these data types. as is the case with conventional SQL relations.

To facilitate BLOBs and ADTs, SQL's CREATE will need to be expanded. Abstract data types may specify attributes and methods (operations) in the member list. An abbreviated version of the syntax follows:

CREATE TYPE <ADT name> [<member list>]
or
CREATE BLOB <BLB name> [<method list>]

These declarations can then be used by the standard SQL operators to create tables of ADTs or BLOBs, to select an ADT or BLOB from a table of items, to update or delete an ADT or BLOB, to insert a new ADT or BLOB in the table, or to perform defined applications operations on an ADT or BLOB. SQL3 requires less alteration due to its support for ADT within an object's specification (Gall92).

3.1.2 Data Constraint Definitions

Additional data definition constructs are required to provide for constraint definitions on data items for:

- temporal constraints
- spatial constraints
- data dependency constraints
- storage constraints
- access constraints

These constraints are used to define the boundaries upon which data stored within the database can be considered consistent. The conventional range, boundary and type constraints are extended for time, space, storage and dependencies as itemized above. Temporal constraints are used to specify within what range of time a data item is considered correct and consistent. For example, during the definition of a data item, we could specify a constraint to indicate that the data item must be updated every *n* seconds beginning at some initial time until some future time as:

[UPDATE EVERY <interval>
  [AFTER <time exp>] UNTIL <time exp>]

if the condition is violated the data item is considered temporally inconsistent. Conditionals such as BEFORE, AFTER, UNTIL, WHEN, BY, and EVERY are needed to provide flexibility in defining timing constraints on data definitions and use. Also, SQL2 does not allow constraints to include references to any of the functions which return dates and times (DD92). This restriction must be relaxed.

Likewise, during the specification of a table, we could define a spatial constraint:

[SIZE UPPER LIMIT <integer>]

indicating that this is the maximum number of data items that can be in this table. This would allow a real-time programmer, for example, to determine an upper bound on accessing the entire table.

Data dependency constraints can be specified through the use of the CREATE TRIGGER statement proposed for SQL3 (MS92). Here is an abbreviated form of the statement:

CREATE TRIGGER <trigger name>
  (AFTER | BEFORE)
  (UPDATE [OF <data item>] | INSERT | DELETE)
  ON <table> <action>

Hence, the trigger can be used to specify an action to be taken before or after a given event (such as an update of a data item). The action may update information in other tables, or could be used to signal an exception.

Storage constraints are used to allow programmers the ability to specify where and how a data item or table is to be stored. This feature is essential information for real-time applications to bound search, execution time and access time for the specified data structure. The following clause could be part of a table definition:

[STORE IN <storage type> [AT <location>]]

This clause could be used to specify that the table be stored in main memory at a particular memory location. Other possible constraints may deal with specifying how to translate a data item from one form to another or if this is allowed. Constraints on access privileges can also be specified. We will look at these in the data control language section of this paper.

3.2 Data Control Language Needs

To support added security and fault tolerance requirements (Gord93) the data control language will need additional features. Security extensions will require the ability to specify how data items can be used. For example we may wish to limit the access to a data item to just the owner, or to some subset of applications. Declarations such as *public, private, protected, owner, group, and use rights* will be needed to spe-

cify how a data item is controlled. Further details of security requirements and possible language extensions can be found in the NGCR DISWG requirements document (Gord93) and its references. Further data control language extensions may be needed to specify forms of recovery to guarantee security considerations. A discussions of these concepts is beyond the scope of this paper.

3.3 Data Manipulation Language Needs

The greatest volume of added features will be found in the data manipulation language. Added services for scheduling, recovery, concurrency control and constraint enforcement will be required. In the following paragraphs we will address some of the possible extensions to SQL.

3.3.1 Transaction Structural Specification

Transactions need to be defined and delineated so that pre- optimizations can be performed. In addition due to the time constrained and concurrency requirements of numerous real- time applications, transactions must take on different forms and characteristics than the conventional monolithic forms provided. The transactions basic structure should mirror the programming language structure it is hosted on. For example if an object oriented programming environment is used, then the transactions should be of the form of actions on objects.

In addition, transaction definitions should be flexible enough to allow for defining partitioned transaction structures, nested transactions structures or possibly projected transaction structures (BHG87,Fort93). The description of a transaction should include a specification and a body. The specification can be used to define the characteristics of the transaction, as well as the structure of the subtransactions (subtransactions are defined in the same manner as transactions).

A second place where characteristics of a transaction can be specified is the SET TRANSACTION statement. In SQL2, this statement can be used to specify characteristics of the next transaction. RT-SQL needs to extend the SET TRANSACTION statement to include transaction and subtransaction identifiers, to allow specification of these characteristics at any time. As in SQL2, transaction characteristics would include *access mode*, *diagnostics area size* and *isolation level*. Within a transaction specification, the definition of the subtransaction structure may be done using the following declarations:

[PARTITION <partitioning spec>]
[NEST <nesting spec>]
[PROJECT <project spec>]

These constructs could be used to specify portions of a transaction that can be done concurrently or in parallel, to specify transaction correctness criteria based on correct execution of partitions, to define recovery for each partition separately, to define commit criteria for each partition separately, to specify separate concurrency control protocols for each partition, or to specify rules for concurrent execution separately. Some of these uses will be shown in later subsections.

Additional control structures as specified in the SQL-3 (Gall92) evolving standard need to be added to the RT-SQL specification. For example the ability to specify a sequence of SQL statements in a procedure or block, to provide flow of control statements such as looping, branching, case, and conditionals. Also, the ability to specify exception handling facilities within the database application's code.

There may be a need to specify the relative criticality of a transaction or portion of a transaction. To provide this RT- SQL requires the transaction specification to allow definition of the criticality of a transaction as follows:

[CRITICALITY <criticality level>]

This clause could also be specified on in the SET TRANSACTION statement as well as on an individual SQL statement or on a block of statements.

3.3.2 Temporal Constraints

Time management is one of the most important features needed to realize real-time database management. Transactions must have the capability to be scheduled based on time synchronization, event synchronization or constraint requirements.

Real-time transactions can be synchronous or asynchronous. A synchronous transaction requires a start time, period, and possibly end time and conditions. a possible transaction synchronous temporal specification is:

[SYNCHRONOUS
  START <time exp>,
  PERIOD <time interval exp>
  [, DEADLINE <time exp>]
  [, WHEN <conditions>]]

A possible transaction asynchronous temporal specification is:

[ASYNCHRONOUS
  EVENT <event>,
  START <time exp>,
  DELAY <time interval exp>,
  [, DEADLINE <time exp>]
  [, WHEN <conditions>]]

These specifications allow for the specification of time constraints on the execution of transactions.

The time constraints can take differing forms based on the applications needs. Time can be absolute as

in wall clock time, it can be relative to some action or absolute time, it can be an event and time can have conditional qualifiers such as: BEFORE, AFTER, BY, WITHIN, to quantify the severity of the timing requirement. These additional time qualifiers allow for the specification of hard deadlines, firm, soft or no deadlines on transaction execution.

3.3.3 Transaction Properties

Transactions in a real-time system must have the capability to alter the conventional meaning of the ACID properties. For example it may not be necessary nor desirable to require all transactions or portions thereof to be *atomic*, *consistent* or *independent*, but it is desirable for them to be *durable*. In order to support such loosening of conventional transaction ACID properties we need language constructs for each feature.

To guarantee that a transaction or a portion thereof be atomic or not atomic, to allow the alteration of guranteed atomic execution requires a construct such as:

[INITIALLY [NOT] GUARANTEED
  [NOT] CHANGEABLE]

The INITIALLY mode (similar to what can be found in constraint deferrability mode) indicates if the application can change the mode.

This specification can appear in a transaction specification, or within a SET TRANSACTION statement. Transactions (and subtransactions) should have identifiers which can be included within a SET TRANSACTION statement to make this possible. Such a feature would allow the applications designers to determine the granularity of atomicity to be used, instead of having it dictated by serializability.

To allow or block the preemption of a transaction or subtransaction, or to allow the alteration of the constraint at some future time requires a feature such as:

[INITIALLY [NOT] PREEMPTABLE
  [<conditions>] [NOT] CHANGEABLE]

which can appear in a transaction specification, or within a SET TRANSACTION statement. This would provide mechanisms to construct critical sections of database code that cannot be interrupted. The control moves over to the applications designers not the database designers.

To allow or block recovery of a transaction or subtransaction, or to allow the altering of recovery based on some user specified conditions requires a feature such as:

[INITIALLY [NOT] RECOVERABLE
  [<conditions>] [NOT] CHANGEABLE]

which can appear in a transaction specification, or within a SET TRANSACTION statement. This feature would allow the partial or total recovery of a transaction based on the specification of the applications needs, not on the needs of the database or database manager.

To allow concurrent execution of a transaction's statements and subtransactions requires a feature such as:

[<conditions>] BEGIN PARALLEL
  [<statement list>] END PARALLEL

[<conditions>] BEGIN CONCURRENT
  [<statement list>] END CONCURRENT

This is based upon the compound statement proposed for SQL3 (Gall92), and as such should also include a block label, local variable declaration list, and exception handler. The statement list may include SQL statements and subtransactions. These features allow for the specification of concurrent and parallel operation of transaction partitions. Such a feature is essential for real-time applications such as robotics and C^3 systems where tight data and temporal synchronization is required.

3.3.4 Active Triggers

Another important requirement within a real-time database management systems are triggers. A trigger is a means for the database to provide additional information to applications to effect synchronization and correct operations. Data in the database can be set up such that a transaction or action can be started based upon a given condition. The triggering condition can be specified on the data or within transactions themselves. To provide this feature RT-SQL needs the addition of a *trigger* operator. The trigger operator allows for the specification of a trigger and the conditions upon which the trigger becomes active. Recall the trigger notation presented earlier:

CREATE TRIGGER <trigger name>
  (AFTER | BEFORE)
  <event> ON <table> <action>

where event was either UPDATE, INSERT, or DELETE on the named table. The format of the <action> clause includes:

[WHEN '(' <search condition> ')']
  '(' <statement list> ')'

The notion of an event should be expanded to include any known event or exception. For example, the start of particular transaction could be signaled, so that that event could be detected by the trigger statement. Hence, the trigger notation should be generalized as follows:

CREATE TRIGGER <trigger name>
  (AFTER | BEFORE)

<event> <action>

where <event> can be any known event, including updates, deletes, and inserts on tables. The format of the <action> clause includes:

[WHEN <condition list>] '(' <statement list> ')'

The condition list is the set of conditions upon which this trigger tests for activation and the statement list may include operations, transactions, and signaling exceptions.

3.3.5 Transaction Recovery

As in conventional databases real-time database transactions must have features for recovery. Unlike conventional databases real-time database recovery needs to be directed by the applications needs not the database systems. Recovery features should provide means to recover from timing, transaction, software or hardware faults based on an applications semantic requirements.

Recovery can be automatic, semiautomatic or manual. The applications writer should have the ability to select only the recovery needed by his (her) transaction. A SET TRANSACTION statement could include the following recovery specification:

[RECOVER ON <condition>
  [AUTO [<atomic>] |
  [SEMI [<recover-body>] |
  [MANUAL]]

Such a feature provides the applications writer the ability to only recover what is necessary. For example in a real- time tracking system if a track input is missed the application would not wish to go back to a previous state. Instead no recovery is performed, the applications simply delays and waits for the next valid track input.

In such a way the application only needs to pay for the degree of recovery needed, not for the ACID properties full spectrum.

3.4 DBMS Precompilation Needs

A real-time database management system will require some additional support tools in order to generate database structuring and database manipulation code which will operate consistently and correctly. Tools for analysis of a set of generated transactions to determine execution boundaries, to determine and set the locations of data, and the constraints to be adhered to for example must be developed. The scope and contents of these elements of a database management query language are beyond the scope of this paper at this time.

4 CONCLUSIONS and FUTURE RESEARCH

Our reason for developing and presenting these concepts is to stimulate the database and real-time research and development communities to begin addressing the ways in which we can introduce real-time database technology into the main stream of database product developments and database standards evolution. To accomplish this the database community must develop a standard set of language features which augment conventional database management systems languages in such a way that does not limit the use of existing database management systems code, yet allows for the specification of real-time and fault tolerant features required by new applications.

To this end we have proposed a set of high level extensions to the SQL language which will meet some of these needs of applications which use real-time databases. Future work under DISWG and ANSI PRISTG will further these efforts to develop a standard set of features for real-time and predictable database management.

5. REFERENCES

AG88 Abbott R. and H. Garcia-Molina (1988). Scheduling Real-Time Transactions. *Proceedings of ACM SIGMOD Conference*, March, 1988

BHG87 Bernstein P. and V. Hadzilacas and N. Goodman, (1987). Concurrency Control and Recovery in Database Systems, Addison-Wesley Publishing Co, Reading, Ma. 1987.

DD92 Date C. and H. Darwen, (1992). A Guide to SQL Standard, Addison-Wesley Publishing Co., Reading, MA. 1992.

Fish94 Fisher D., (1994), Current Database Standards and Available Technology, NGCR SPAWAR 331-2, 2451 Crystal Dr., Alexandria, Va.22245. February, 1994.

Fort94 Fortier P., (1994). DISWG Database Management Systems Reference Model, NGCR SPAWAR 331-2, 2451 Crystal Dr., Alexandria, Va.22245. February, 1994.

Fort94a Fortier P, (1994). Data Management Concepts for Real-time C^3 Systems , *To Appear in: Proceedings of the Joint Navy IR and IED Symposium*, June, 1994.

Fort94b Fortier P., (1994) ANSI DBSSG PRISTG: Real-Time Database Management Systems Reference Model, *ANSI DBSSG Predictable Real-time Information Systems Task Group, PRISTG Document No. 94-001*, January, 1994.

Fort93 Fortier P.,(1993). D.Sc. Thesis: Early Commit, University of Massachusetts Lowell, April, 1994.

FA94 Fortier P. and M. Audette, (1994). Simulation Analysis of Real-Time Task Scheduling Mechanisms, *Proceedings of the HICSS Conference*, January, 1994.

FS94 Fortier P. and Cdr. G. Sawyer, (1994) DISWG a New Player in NGCR Open Systems Standarads, *to appear in Computer Standards and Interfaces*, 1994.

Gall91 Gallagher L., (1991). Database Management Standards: Status and Applicability, *Computer Standards and Interfaces*, 12, 1991.

Gall92 Gallagher L., (1992). Object SQL: Language Extentions for Object Data Management. *Proceedings of the International Society for Mini and Microcomputers CIKM-92*, 1992.

Gord93 Gordon K., (1993). Requirements for Military Database Management Systems. *NGCR Technical Document No. 010 ver. 1.0*, 15 November, 1992 NGCR SPAWAR 331-2, 2451 Crystal Dr., Alexandria, Va.22245.

Grah93 Graham M., (1993). Real-time Data Management. *IEEE Technical Committee Real-Time Systems Newsletter*, 9(1/2), Spring/Summer, 1993.

HS93 Hamidzadeh B. and S. Shekhar, (1993). A General Framework for Dynamic Scheduling of Real-Time Tasks. *IEEE Technical Committee Real-Time Systems Newsletter*, 9(1/2), Spring/Summer, 1993.

Herl90 Herlihy M (1990). Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types. *ACM Transactions on Database Systems* 15(1), March, 1990.

HW90 Herlihy M. and J. Wing (1990). Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3), July, 1990.

KLS90 Korth H., E. Levy and A. Silberschatz (1990). A formal approach to recovery by compensating transactions. *Proceedings of the 16th Very Large Database Conference*, 1990.

LL88 Lin K. and M. Lin (1988). Enhancing availability in distributed real-time databases. *ACM SIGMOD Record*, 17(1), March 1988.

MS92 Melton J. and A Simon (1992). *Understanding the new SQL: A Complete Guide.* Morgan Kauffman Publishers, SanMateo, Ca., 1992.

Naka93 Nakazato H. (1993). *Issues in Synchronizing and Scheduling Tasks in Real-time Database Systems.* PhD Thesis, University of Illinois at Urbana-Champaingn, Urbana, Il., 1993.

OV91 Ozsu T. and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall Inc., Englewood Cliffs, NJ, 1991.

Son88 Son S. (1988). Real-time Database systems: Issues and approaches. *ACM SIGMOD Record*, 17(1), March, 1988.

SYKI92 Son S, S. Yannopoulos, Y. Kim, and C. Iannacone (1992). Integration of a database system with a real-time kernel for time-critical applications. *Proceedings of the International Conference on Systems Integration*, June, 1992.

SS93 Spuri M. and J. Stankovic (1993). How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Technical Committee Newsletter on Real-time Systems*, 9(1/2), Spring/Summer, 1993.

WCPP93 Wolfe V., L. Cingiser, J. Peckham, and J. Prichard (1993). A model for real-time object oriented databases. *IEEE Technical Committee Newsletter on Real-time Systems*, 9(1/2), Spring/Summer, 1993