Open access • Journal Article • DOI:10.1145/181550.181557

Towards an infrastructure for temporal databases: report of an invitational ARPA/NSF workshop — Source link

Niki Pissinou, Richard T. Snodgrass, Ramez Elmasri, Inderpal Singh Mumick ...+5 more authors

Institutions: University of Arizona, University of Texas at Arlington, Bell Labs, University of Alberta ...+3 more institutions

Published on: 01 Mar 1994 - International Conference on Management of Data

Topics: Electronic mail

Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop

Niki Pissinou, Richard T. Snodgrass, Ramez Elmasri,
Inderpal S. Mumick, M. Tamer Özsu, Barbara Pernici,
Arie Segev, Babis Theodoulidis and Umeshwar Dayal

1 Introduction

Temporal databases has been an active area of research for the last fifteen years, with a corpus nearing 700 papers [Kline93]. Most database conferences include at least one paper on temporal databases (TDB). Temporal databases are now discussed in several undergraduate database textbooks. There are perhaps one hundred researchers actively studying temporal databases.

During that time an astonishing diversity of temporal data models and query languages has arisen. Most applications, whether business, engineering, medical, or scientific, need to store time-varying data.

Surprisingly, in spite of both this substantial activity and this pressing requirements from the user community, there are no widely used commercial temporal database management systems (TDBMS). One view is that there is an embarrassment of riches in the TDB literature: with so many alternative approaches from which to choose, it is safer for a DBMS vendor not to choose than to choose an approach that ultimately yields to a competing alternative. The same phenomenon may be occurring in TDB research. In contrast to the flurry of activity in query languages and data models, there is a dearth of results in temporal database design and temporal query optimization, in part because there is no commonly accepted consensus data model or query language upon which to base research and development. At a more fundamental level, even the terminology is highly nonstandard. As an example, the terms intrinsic time, logical time, real-world time, and valid time have all been used for the core concept of the time at which something happened.

It was decided in early 1992 that a meeting should be held with the objective of identifying a common infrastructure to provide a foundation for implementation and standardization as well as for further research into temporal databases. Subsequently, on June 14–16, 1993, the International Workshop on an Infrastructure for Temporal Databases was held in Arlington, Texas.

The workshop consisted of plenary sessions (invited talks one day and discussions the other two days) as well as group sessions where four working groups addressed specific issues amenable to infrastructure.

There was also substantial effort both before the meeting, to prepare infrastructure proposals for debate, and after the meeting, to build on the insights that emerged from the discussion. Specifically, an initial glossary of temporal database concepts and a test suite of temporal queries were distributed before the workshop. Both of these documents were amended based on the analysis and critique of the workshop. A language design committee was constituted after the workshop to develop a consensus temporal query language extension to SQL-92; this design also benefited from the discussion at the workshop.

This report documents the discussions and consensus reached at the workshop. The report reflects the conclusions reached at the workshop in June, 1993 and further discussions amongst the group participants through electronic mail. In preparing this report, each group coordinator assembled ideas and prepared an initial draft, which was then reviewed by all the workshop participants.

The record of the deliberations of these four groups, in the following four sections, forms the bulk of this report. Each of these sections begins with the group's charter and a brief snapshot of the status of the field and ends with a list of follow-on efforts. The last section identifies the workshop participants. The full report[1] provides more discussion and many additional references to the literature.

2 Group A: Special Requirements and Approaches

2.1 Introduction and Group Charter

Temporal databases can be used in a variety of applications. In addition to conventional applications handling administrative data of various types, other applications such as of logistics, scientific applications, and artificial intelligence present new requirements to the temporal database community.

The working group on Special Requirements and Approaches, consisting of G. Ariav, M. Baudinet, M. Boddy, C. Dyreson, M. Egenhofer, P. Hayes, F. Olken, B. Pernici, and S. Sripada, with the collaboration of occasional visitors, discussed characteristics of the different application areas, attempted to relate terminology differing from that generally used in the temporal database community, and strived

[1] The full version of this report is available as Technical Report 94-01, Computer Science Department, University of Arizona, January, 1994, 55 pages.

to identify essential features to be provided by a temporal database system in order to be usable by these different communities as well as features and functionality for which additional research is needed.

In the following, the basic characteristics of the different application areas are briefly outlined, followed by an analysis of features to be supported by a temporal database management system (TDBMS).

2.2 Current Status of the Field

G. Ariav started the discussion with a presentation based on a set of problems from case studies illustrated in his position paper at the workshop [Ariav93]. Other application domains were also considered, specifically geographical information systems (GIS), scientific applications, and planning and scheduling applications.

The group isolated two needs not met in current TDBMS proposals: multiple time lines (several validity times seem to be needed), and an undo operation of less global impact than a rollback operation.

A common feature of applications is that temporal data are mostly imprecise and concern relative times. Ordering relationships between events are often more frequent than a precise location on the time axis of the events. An additional common feature, is that there is a need for merging different databases, possibly defined on different time granularities.

2.3 Features to be Supported by a TDBMS

2.3.1 General Discussion

In general, there seems to be a gap between the goals assumed by the temporal database community and the needs discussed in the working group on special requirements.

To define the needed requirements, the first, important consideration is the definition of the boundary between the functionality to be supported by the temporal database system and that provided by applications working with the data stored in the database.

It initially seems desirable that the TDBMS provide not only basic functions, but also advanced features useful for specific applications. For instance, a classical problem found in scientific and logistic applications is the "shortest path" problem. In practice, it seems that in order to provide reasonably efficient access to data, it might be important to provide some support also for this type of computation. A similar related problem is that of providing recursive queries support in relational databases.

However, the group, decided that, at least for the short term, only basic functionality should be considered for the purpose of establishing an infrastructure, provided that the system does not become an obstacle for users to retrieve their data.

An important consideration concerns the nature of temporal data in a temporal database. The request is for particular forms of support for data defined as temporal. This support should be analogous to that provided in classical databases for predefined attribute domains. For instance, if an attribute is defined on the integer domain, it is not possible to insert characters as values. Obviously, more sophisticated types of consistency checks can be defined. Analogously, there is a need for providing true temporal attributes with appropriate consistency checks, to justify the definition of a temporal attribute. For instance, let us suppose that Name functionally determines Salary. The TDBMS should ensure that the same nameis not associated with tow different salaries at the same time. Such integrity constraints may be defined on attribute values that have been specified as valid time attributes. If the temporal attribute is treated simply as any other attribute, there seems to be no need to define it specifically as "temporal."

2.3.2 Basic Functionality

A fundamental need of most users is for support of time values at many different granularities. Appropriate operations must be supplied to perform conversions of time values between the different granularities, and to formulate queries and present results in an appropriate form.

A related feature is the need for a *merge* operation in order to be able to work with data coming from different databases (or relations) defined at different granularity levels.

Concerning times, in scientific databases and in planning and scheduling databases it is essential to provide support not only for times based on the time line (absolute times) but also for time which are relative to other times. To this purpose, the use of *time variables*, both for time points and for time entities, has been suggested (alternative name: symbolic time points/interval).

Accordingly, appropriate temporal relationships (possibly imprecise) have to be defined and supported in the TDBMS (point-point relations, interval-point relations, interval-interval relations). However, only storage support should be provided: reasoning on these times is considered outside the scope of the temporal database.

Several features related to queries have been identified: support for relative times (temporally ordered attributes), support for aggregation operators over time, and support for temporal joins.

In our discussions, the term "relative time" was used in several senses: times specified with respect to an unanchored rather than an anchored ordering, but also times that were "variable" (e.g., we would consider "Easter" as a variable time, dependent on context for disambiguation/grounding, and "after A but before B" as a relative time specified as a position in an ordering that is unanchored to a time-line);

The issue of query result presentation is also important, in particular concerning the presentation of approximate answers, and in providing answers sorted according to a specified time dimensions, to increase readability (re-arranging in time).

2.3.3 Additional Needs

A number of possible extensions to the above basic functionality can be considered: definition of time-varying data types (with interpolation functions, with associated probabilities, and the like) and of associated operations; definition of composite events; associating disjoint intervals with data values; and support of periodic data.

2.4 Conclusion and Follow-on Efforts

The result of this working group can be summarized by considering the following question: "Can the diverse needs of the user community be served by temporal database technology?"

Characterizing the commonalities of the user community is an enterprise doomed to fail, because users are many and their needs are diverse. Nonetheless, it is an effort we attempted in our group, gathering potential users from a variety of disciplines. Although the group was not representative of the user community as a whole, those that were present were knowledgeable potential users familiar with, by and large, database technology, and their target applications also involved time at a fundamental level. Yet these were only *potential* users of temporal database technology, primarily because of two factors.

First, no common infrastructure for temporal database research exists. This lack of common infrastructure is detrimental, not only from a research perspective, but more importantly for this group, from a pure salesmanship perspective. Users could not say what a temporal database is, nor even begin to comprehend how it could be of service to their applications. Towards this end, the glossary was important, yet at the same time confusing. The glossary was couched in the language of temporal database researchers. But as researchers in other disciplines have their own (implicit) glossaries for time related concepts; the "conceptual gap" between the two glossaries was difficult to bridge. Also, the lack of an infrastructure document led users to look for such in the glossary, but the glossary was not written nor designed for such a purpose. Consequently, basic concepts such as chronon and event remained permanently baffling, primarily because there exists no "road map" to provide users an understanding of how these terms fit together.

This raises the issue of whether the glossary should serve as a document for researchers active in the field or provide a gentle introduction to temporal databases to researchers and users from other communities. The consensus of the group seemed to be that the glossary can only be written for researchers in the field and that some other form or document should present the infrastructure and advertise the utility of temporal databases. The glossary should provide a backdrop to the infrastructure, only giving meaning to words that are unfamiliar to the reader.

The second factor is related to the first. The users in our group have developed tools to meet their needs (e.g., the geographer in our group developed a GIS). By and large, users want to "extend" their tools to include support for time values. The key characteristic of this extension is its ad-hoc nature. The tools exist and a great deal of effort and research has been invested in creating them. By and large these users are only interested in providing better support for time values or time-related processes, rather than replacing these tools with a temporal database. In order for temporal database technology to serve these users, that technology must provide a platform on which these tools can reside, without requiring substantial modification of the tools themselves. Perhaps one could characterize this need by saying that users are very strongly in favor of an "open" architecture.

Because time is considered an "add-on" in these systems, many of the issues discussed by other working groups for inclusion in the infrastructure did not emerge as user concerns. In particular, we did not discuss nor even raise questions as to whether SQL-92 or SQL3 should serve as a platform for an infrastructure query language and data model. Users have their own "high-level" query language targeted for their applications (e.g., temporal reasoner, GIS, human genome project) and any language that is capable of extracting time-related information (in a very primitive way) would suffice. This is not a criticism of the laudable goals of these other groups, only an observation that none of the users in our group currently base their tools on SQL-like interfaces or databases. The consensus TSQL2/3 effort is certainly of importance to the large community of actual database users. But almost unanimously, the users in our group were uninterested in the differences between these alternatives because they already have their own query languages and data models. It is an open question as to whether the users wanted a temporal abstract data type (ADT) or something more complex.

In essence, it is a matter of timing. The temporal database community is somewhat late to the game due to a lack of common infrastructure and working temporal databases. Consequently other players have already taken the field. If we are to have any impact on the game, as a practical matter, the question of how to integrate with existing tools is of primary importance and runs deeper than SQL-integration.

Some specific user needs did emerge, aside from the "open" architecture requirement. Let us consider relative time. Can it be supported by a temporal database? That depends on what is meant by "supported". Certainly, a temporal database can store such times and their associated constraints. Interpreting these constraints however is another matter, and lies in the sphere of general temporal reasoners rather than temporal databases. But embedding a temporal database within a temporal reasoner is exactly what the users in our group desired. Our users had temporal reasoners. They were interested in knowing whether and how their reasoners could be seamlessly coupled with a temporal database. They did not expect the temporal database to interpret the relative times, that would be done by the reasoner. At first glance, it would seem that the only requirement for a temporal database to "support" relative times is the capability of storing such times (as valid times) and passing "uninterpreted" times to a higher level. Since these times are uninterpreted, they should remain inert in determining temporal keys and normal forms.

This is not to suggest that all user requirements can be accommodated as easily; a user, say, who desires continuous times may be somewhat harder to satisfy. Rather it suggests that the functionality of a temporal database must be clearly and distinctly articulated before integration can take place. The above example places relative times outside the sphere of temporal databases (a widely-held viewpoint within the community?). The line must be drawn everywhere on exactly what is and what is not supported by a temporal database, perhaps further dividing that support into core and optional functionality. The consensus of our group was that user needs are diverse, consequently core functionality should be minimized and ease of extensibility should be maximized.

3 Group B: Extending SQL-92

3.1 Introduction and Group Charter

The working group, consisting of I. Ahn, J. Clifford, F. Grandi, C.S. Jensen, W. Käfer, K. Kulkarni, N. Lorentzos, R. Snodgrass, A. Tansel, with occasional visitors, addressed a fairly narrow but complex topic: how should SQL-92 be extended to support time in a comprehensive fashion. The ultimate goal is to produce a concrete language definition that can be used by temporal DBMS researchers as an infrastructure, incorporated into legacy (relational) DBMS products, and considered by the SQL standards committee. The (quite ambitious) goals of this working group were to put into place a structure for such a language definition, and to reduce the number of possibilities to a small set that can be further evaluated in the coming months.

3.2 Current Status of the Field

Researchers have been prolific in developing temporal data models and query languages, in an attempt to find the right tradeoffs among a set of irreconcilable constraints [MS91a]. Over the last fifteen years of work, a total of over two dozen temporal extensions of the relational data model have been proposed. Approximately half of these models support only valid time; three models support only transaction time; and the remaining seven or so support bitemporal relations. The temporal data models may be compared by asking four basic questions: how is valid time represented (alternatives include event, interval or temporal element stamping of individual attributes or tuples), how is transaction time represented (alternatives include event, interval, three events, or temporal element stamping of individual attributes, tuples, or sets of tuples), how are attribute values represented (alternatives range from atomic valued, to ordered pairs, to triplet valued, to set-triplet valued), and is the model homogeneous and coalesced (all four alternatives are represented).

Most temporal data models are paired with a temporal query language proposal. Some two dozen temporal relational query languages have been proposed, including seven extending the relational algebra, five extending Quel, seven extending SQL, and a few being based on other formalisms.

Support for time in conventional data base systems (e.g., [TC83, OC87]) is entirely at the level of user-defined time (i.e., attribute values drawn from a temporal domain). These implementations are limited in scope and are, in general, unsystematic in their design [Date88, DW90]. The standards bodies (e.g., ANSI) are somewhat behind the curve, in that SQL includes no time support. Date and time support very similar to that in DB2 is included in the SQL-92 standard [MS93]. SQL-92 corrects some of the inconsistencies in the time support provided by DB2 but inherits its basic design limitations [SS92]. The SQL3 draft proposal contains no additional temporal support over SQL-92.

3.3 Level of Language Support

The primary realization to come out of the workshop was that there were three fundamental viewpoints on how time should be incorporated into SQL. In the following, we present each of these viewpoints, along with some of their supporting arguments.

The first viewpoint argues that the SQL data model is already quite close to having the support required by temporal applications. The additional support that is necessary is primarily in the algebraic operators and to the syntax of the language. A concrete realization of this viewpoint is the IXSQL proposal, which extends SQL with a *generic interval* data type (of course, the focus here is on intervals of *time*). The data model is identical to that of SQL, with the addition of `DATEINTERVAL`. The algebra for this language is an extension of the relational algebra, retaining the traditional operators in an unmodified form, and adding two new operators. **Unfold** converts an interval into a set of time points, with the remaining attributes duplicated for each time point. **Fold** is the inverse operator. In terms of the SQL syntax, new predicates on intervals are defined, and two clauses are added, a `REFORMAT` clause, to support **Fold** and **Unfold**, and a `NORMALISE` clause, which can be simulated with the `REFORMAT` clause.

Several advantages accrue from this approach.

- Since intervals are generic, and can thus be defined over any metric, this approach naturally supports spatial and spatiotemporal databases.
- Since the extensions to SQL are minimal, especially compared with other approaches, implementation is less difficult, and acceptance by the user community may be easier to attain.
- Multiple time (and other metric) intervals may easily be incorporated.
- Every snapshot relation is also a valid valid-time relation.

The second and third viewpoints share the belief that time is a basic aspect of data, and therefore should be incorporated in a fundamental way into the data model and query language. The two viewpoints differ on the timing of the language definitions. The second viewpoint holds that, with SQL-92 an accepted standard and SQL3 being actively designed, there is no sense in extending SQL-92. Instead, efforts should be directed towards adding time to the SQL3 proposal, yielding perhaps a temporal query language standard in the 1995–1996 time frame.

Several advantages have been stated of a single SQL3 extension.

- SQL-92 is frozen, so any extensions based on SQL-92 will be rendered meaningless when SQL3 is accepted.
- SQL3 has several data modeling constructs, including object orientation, which can aid in the development of temporal extensions. For example, it allows nested relations to be simulated, thereby accommodating more temporal data models than SQL-92.
- A two-pronged approach would be difficult to coordinate, and could easily result in incompatible proposals.
- Research is active in temporal databases, with new ideas appearing all the time. It is important to do the design "right," or we will be saddled with a poor design, with no opportunity to change it (we get only one chance).

The third viewpoint favors a two-pronged approach, in which parallel efforts would consider adding time to SQL-92 and to SQL3. The rationale is that time will be added to the SQL standard only when there is implementation experience available, and that won't occur unless there is a consensus extension of SQL that admits a straightforward implementation without requiring SQL3 constructs.

Proponents of the third viewpoint counter with advantages of their approach.

- SQL-92 provides a stable basis on which to do language design; SQL3 is constantly changing.
- Designing an SQL3 extension will not impact actual applications before 1997, when the first implementations supporting SQL3 may start to appear.
- Should extension of only SQL3 proceed, there is the chance that a vendor will go ahead and implement temporal support now, in an ad hoc fashion, which the SQL3 standard will be forced to incorporate (as happened with user-defined time).
- The SQL3 standards bodies will not be interested in fundamental time support until users clamor for it, and until a commercial, relational DBMS (or perhaps two such systems) supports time.

These three viewpoints are clearly in conflict. However, each has its vocal proponents, marshaling strong technical arguments to advocate their position. The disparity between these distinct viewpoints offers one explanation as to why there has not been greater consensus in the field. Clearly, it is difficult to arrive at a single data model when there are fundamental disagreements concerning even the extent to which time should be incorporated in the model.

3.4 Desired Functionality

Much of the discussion of the working group was devoted to determining the functionality that is desired in a temporal query language. We now list the aspects discussed. We refer to an extension of SQL (-92 or 3) as *TSQL*, for convenience, keeping in mind the diversity of opinion listed in the previous section.

There have been three types of time that may be used in a temporal database: *user-defined time*, *valid time*, and *transaction time* [SA86]. User-defined time has garnered support in most commercial DBMS's (and is present in the SQL-92 standard), and transaction time is supported in some object-oriented databases (as version identifiers) and one relational database (Montage). However, the range of applications that could use valid-time support (most applications, in fact), as well as those that could use support of all three kinds of time (which is not the majority of applications, but certainly a sizable portion), dictates that a bitemporal extension of SQL is warranted.

At the same time, it is important to support legacy applications. Hence, temporal support should be optional in both the schema and in the query language. This requirement translates into the ability to specify snapshot relations (for which no temporal support is required), as well as valid-time, transaction-time, and bitemporal relations in the `CREATE TABLE` statement. It also implies that queries should be able to include multiple types of relations in the `FROM` clause, and evaluate to multiple types of relations. For example, it should be possible to compute via `SELECT` a snapshot relation from a bitemporal relation.

The extension to SQL should be upward compatible. Existing SQL queries should remain valid in TSQL. Query language reducibility is also important: an SQL query, evaluated on a temporal database as a TSQL query, should result in a temporal relation, which, when timesliced at a particular time, yields the same snapshot relation that results when this same query is evaluated as an SQL query on the timeslice of the temporal database at the same time. This property ensures that user's intuition concerning SQL will transfer over wholesale to TSQL.

Some data model proposals require that the underlying valid time domain extend only to now. Other data models support future time. (Note that transaction time, on the other hand, is never allowed to extend past now: it is impossible to know entirely accurately what will be stored in the future in the database). Planning applications require future time support; hence, it should be present in the data model for TSQL.

The controversy of discrete versus continuous time surfaced in several working group discussions, as well as the workshop plenary sessions. As this topic has been discussed for literally thousands of years by philosophers, mathematicians and physicists, it is understandable that the intricacies of this dichotomy would not be fully resolved in this workshop. Nevertheless, this working group agreed that the *representation* (as opposed to the conceptual model) should be discrete. Gio Wiederhold invoked a useful analogy of real numbers and their representations. While floating point numbers in computer programs can be conceptualized by programmers as real (i.e., continuous) numbers, their representation must necessarily be discrete. The same should hold for timestamps in TSQL's data model.

A more restricted incarnation of this issue is the distinction between open and closed intervals. An *open interval*, generally denoted as $[a,\ b)$, where a and b are timestamps, contains the time between a and b, as well as the time instant a, but *not* the time instant b. Conversely, the *closed interval* $[a,\ b]$ contains the instant b. In a discrete representation, $[a,\ b+1) \equiv [a,\ b]$; in a continuous model of time, there is no successor to b, and so the two are not comparable. At the representation level of TSQL, which uses discrete time, the distinction is not important. At the language level, which the user can pretend is based on continuous time, the language should support both open and closed intervals in the presentation (input and output) of temporal values.

The final issue discussed at length was that of *ungrouped* versus *grouped completeness*. These terms were presented by James Clifford at the workshop, based upon his previous research [CCT93]. In this work the authors attempt to contrast those models which employ tuple-time-stamping, which they term temporally ungrouped, and those which employ complex attribute values bearing the temporal dimension, which they term temporally grouped. After defining canonical versions of these two types of data models, called M_{TU} and M_{TG}, respectively, they present logic-based query languages for each of them and propose them as standards for measuring the expressive power of query languages for such models. They further demonstrate that the grouped models are more expressive than the ungrouped models, but define a precise, though cumbersome, technique for extending a temporally ungrouped model, by means of a group surrogate, in such a way as to extend its expressive power to that of the temporally grouped complete models. In surveying some (but by no means all) of the models that have appeared in the literature, they demonstrate the following: (i) several algebras and calculus-based query languages are ungrouped complete, (ii) the calculus L_h is, by their definition, grouped complete, and (iii) to their knowledge no algebra has been shown to be grouped complete.

While the expressive power of ungrouped complete was generally accepted as a desirable property for TSQL, there was considerable discussion concerning grouped complete. The benefit of grouped complete is that it supports a rather strong notion of the "history of an attribute," called a *history* in the Glossary. For example, one can talk about "John's salary history" as a single object, and ask to see it, or define constraints over it, etc. If the data model and query language are not grouped complete, then the salary history will be lost unless the key (here, the name) is always retained, which places a burden on the user. Ungrouped models also generally require some kind of time-invariant key to identify entities in the miniworld being modeled by the database, whereas "histories" are supported directly in grouped models without any need for time-invariant keys.

The primary concern raised by some members of the working group was one of implementability. It was pointed out that no implementation of a grouped model exists, but this was countered by the observation that few of the proposed models of any ilk have been implemented. A formal mapping of a grouped complete data model onto an ungrouped complete model, via system-maintained surrogates, has been given. However, it was pointed out that this mapping has never been implemented, and the concern was raised that implementing joins in this approach appears to some to be difficult.

The working group members fell into two camps. One position was that the lack of an existing SQL extension definition that was grouped complete, as well as the lack of any implementation experience with grouped complete data models, rendered this requirement of grouped complete too risky to incorporate into TSQL at this time. The other position held that the ultimate aim was to make life easier for the user, even if it complicated the implementation, and thus grouped complete should be a requirement of the model.

This discussion can be examined in light of the three viewpoints presented earlier. The first viewpoint, minimally adapt the data model to support time, favors neither ungrouped nor grouped complete, as both of these distort the original relational model to too great a degree. The second viewpoint, that only SQL3 should be extended, is comfortable with grouped complete. The third viewpoint, advocating definition of both TSQL2 and TSQL3, was generally comfortable with TSQL3 being grouped complete, but not so with TSQL2.

3.5 Separation of Concerns

As previously mentioned, there are now over two dozen temporal data models, each with one or more associated query languages. While such a diversity of approaches is a reflection of the excitement and ferment in the area of temporal databases, it also at some point may become counterproductive.

Focusing on data *semantics* (what is the meaning of the data stored in the data model), data *presentation* (how temporal data is displayed to the user), on data *storage* (what regular storage structures can be employed with temporal data), and on efficient *query evaluation*, has complicated the primary task of capturing the time-varying semantics. The result has been a plethora of incompatible data models and query languages, and a corresponding dearth of database design and implementation strategies that may be employed across these models.

The previously proposed data models arose from several considerations. They were all extensions of the conventional relational model that attempted to capture the time-varying semantics of both the enterprise being modeled and the state of the database. They attempted to retain the simplicity of the relational model; the tuple-timestamping models were perhaps most successful in this regard. They attempted to present all the information concerning an object in one tuple; the attribute-value timestamped models were perhaps best at that. And they attempted to ensure ease of implementation and query evaluation efficiency; the backlog representation may be advantageous here.

Most proposed models aim at being suitable for data presentation, for data storage, and for capturing the temporal semantics of data. Seen solely as means of capturing the temporal semantics, such models exhibit presentational and representational anomalies because they encode the temporal semantics in ways that are more complicated than necessary. Put differently, the time-varying semantics is obscured in the representation schemes by other considerations of presentation and implementation.

It is clear from the large number of proposed data models that meeting all goals simultaneously is a difficult, if not impossible, task. We therefore advocate a separation of concerns, i.e., adopting a very simple *conceptual* data model that captures the essential semantics of time-varying relations, but has no illusions of being suitable for presentation, storage, or query evaluation. Proposals for this conceptual data model were discussed, but a final choice was not made.

Figure 1 places the conceptual temporal data model with respect to the tasks of logical and physical database design, storage representation, query optimization, and display. As the figure shows, logical database design produces the conceptual relation schemas, which are then refined into relation schemas in some *representational* data model(s) during physical database design. The query language itself would be based on the conceptual data model. Query optimization may be performed on the logical algebra, parameterized by the cost models of the representation(s) chosen for the stored data, and in the algebra of the representational model. Finally, display presentation should be decoupled from the storage representation, and should be capable of exploiting the several existing data models having convenient display formats.

Note that this arrangement hinges on the semantic equivalence of the various data models. It must be possible to map between the conceptual model and the various representational models. An appropriate conceptual data model would allow equivalences to be demonstrated with many of the representational models thus far proposed. This equivalence should be based on *snapshot equivalence*, which says that two relation instances are equivalent if all their snapshots, taken at all times (valid and transaction), are identical. Snapshot equivalence provides one means of comparing rather disparate representations. However, it can be demonstrated that a grouped relation can be snapshot equivalent to a large number of ungrouped relations, only one of which carries the same information content. Some argued that the notion of strong equivalence [CCT93], somewhat (but not entirely) captured by the the term "history equivalence" in the glossary, provides a more appropriate means of comparing disparate representations.

3.6 Conclusion and Follow-on Efforts

As mentioned in Section 3.3, there were conflicting viewpoints on a temporal extension of SQL. They can be summarized as (a) with the addition of an interval data type, there will be sufficient support in SQL2/3's data model to support applications using temporal data; (b) a two-pronged effort should be initiated, the first being a short-term effort to define a temporal extension to SQL-92 and the second being a long-term effort to define a comprehensive extension to SQL3, and (c) temporal support should be added, but only SQL3 should be extended. Whatever the approach, it was agreed that the temporal data model underlying the language be designed solely in terms of its semantic properties, with distinct and possibly multiple data models being employed for representation and presentation.

4 Group C: Advanced Temporal Databases

4.1 Introduction and Group Charter

The overall objective of the discussions in working group C, consisting of A. Buchmann, S. Chakravarthy, T.-S. Cheng, K. Dittrich, S.K. Gadia, T. Lawson, I.S. Mumick, M.T. Özsu, N. Pissinou, K. Ramamritham, A. Segev, M. Soo, S. Su, B. Theodoulidis, and G. Wuu, was to identify the common infrastructure for the next generation of temporal database concepts including extensions of the relational data models as well as the adoption of concepts from the semantic and object-oriented data models.

While it is true that the majority of the work on temporal databases has been in the context of the relational data model, a number of approaches based on semantic data models, such as the entity-relationship, infological and object data models, have appeared in the literature. The motivation behind all of these approaches is that the relational data model is considered to be insufficiently expressive for complex database applications such as multimedia, executive information systems, computer-aided design (CAD), computer integrated manufacturing (CIM), and geographical information systems (GIS). These applications have strong requirements to model the temporal or spatio-temporal relationships between objects. Therefore, "temporality" is an important (even if not integral) part of the next generation of database systems. Another trend that started in the 1980's is the incorporation of constraints, triggers, and rules in relational and object-oriented databases. Work in this area is concerned with active temporal databases and was considered at the group discussions.

In view of this, the overall objective of the discussions in group C was to identify a common infrastructure for the next generation of temporal databases, including extensions to the relational data model and object-based models. These issues were discussed in two subgroups, then integrated in plenary sessions of group C. Subgroup C1, consisting of S.K. Gadia, T. Lawson, M.T. Öszu, N. Pissinou, S. Su, B. Theodoulidis and G. Wuu, addressed data modeling concepts, time concepts and the incorporation of time into the next generation of temporal databases, with an emphasis on object based models. Subgroup C2, consisting of A. Buchmann, S. Chakravarthy, K. Dittrich, I.S. Mumick, K. Ramamritham, and A. Segev addressed the area of active temporal databases with particular reference to the notion of temporal rules.

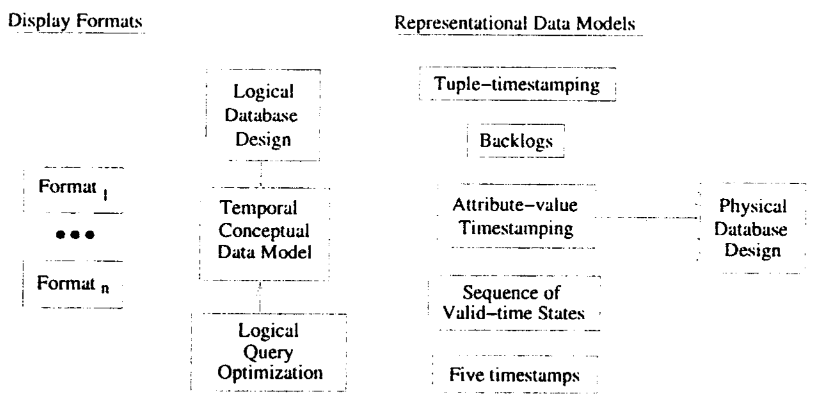
Display Formats
Representational Data Models
Logical Database Design
Format 1
•••
Format n
Temporal Conceptual Data Model
Logical Query Optimization
Tuple-timestamping
Backlogs
Attribute-value Timestamping
Sequence of Valid-time States
Five timestamps
Physical Database Design

Figure 1: Interaction of Conceptual and Representational Data Models

The following sections elaborate on the consensus reached for the infrastructure and the open issues and future work that need to be carried out in order to complete the infrastructure for the next generation of temporal databases.

4.2 Current Status of the Field

4.2.1 Temporal Object Based Modeling

The two most prominent models that provided the basis for the development of temporal conceptual models are the entity-relationship (ER) model [Chen76] and the object based model. The ER model deals with the structural component and is founded on the notions of entity and relationship. The object model deals with both the structural and behavioral components and is founded on the notions of object, structure and behavior (method). Furthermore, a number of approaches include notions like Event-Condition-Action (ECA) rules that deal with the constraint component. Several approaches of introducing time into an object based data model were discussed. The group isolated three main approaches: (1) to extend the semantics of a preexisting snapshot model to incorporate time directly (built-in); (2) to base the new model on a snapshot model with time appearing as an additional attribute(s); and (3) to move in an independent direction, developing entirely new approaches.

4.2.2 Active & Real-Time Databases

Active databases evaluate conditions and execute actions in response to event occurrences (either primitive or complex) according to the semantics of rule processing in active databases. Incorporation of active capability has typically been addressed with respect to a snapshot (i.e., non-temporal) database. A limited notion of time is used in events (e.g., temporal events and composite events) and for specifying deadlines (e.g., complete action prior to a given time). A large body of work exists on the specification of rules, its execution semantics, modeling of events, and incorporating active capability into object-oriented paradigms. [Cha92, DBB+88, MD89, WF90]

Although rules have been used in temporal databases, there is no agreement on when a rule itself, used in a temporal or non-temporal database, may be considered to be temporal. A related issue is, when does a rule require temporal support for its activation? Further, rules often are not modeled in the same way as data; rules should be treated as first class objects, and so rules must be subject to the same temporal semantics as data. The working group addressed this new aspect of temporal rules in addition to defining rule structure for active databases.

Work on integrating temporal and active database features has started appearing in the literature only recently, e.g., [GJMS93]. Some related issues have also been discussed in the context of real-time databases [BB93, Rama93a].

4.3 Next Generation Temporal Data Modeling Concepts

The purpose of this discussion was to identify the key concepts of the next generation of temporal data models and languages. It was agreed that the next generation of temporal data models should be an extended model, rather than extensible with respect to the current generation. This implies the design of the next generation of temporal databases should not be limited to current solutions and approaches to temporal modeling, nor should be an "extensible relational approach."

An important issue, rising from the isolation of the three approaches for next generation temporal data modeling proposed in Section 4.2, concerns the role of a temporal data model. Without clarifying this issue, it is difficult to extend the object model to include temporality. The role of a temporal data model is to visualize and structure temporal data, temporal information and the temporal relationship of objects. In general, one of the main ways of structuring and visualizing temporal data is through the use of abstraction at various levels of granularity. To do so, a temporal model can be discussed in terms of three distinct parts: structures, operations and constraints. The structural component of a data model, deals with objects and their relationships while the operational/behavioral component deals with the manipulation of objects. The constraint component deals with rules for the integrity of the object structure and manipulation over time.

In line with existing data modeling design principles, the basic concepts and components identified by the group for the next generation of temporal data models were classified into three broad categories: Temporal Structural properties, Temporal Operational/behavioral properties, and Temporal Constraint properties. Temporal structural properties describe the objects of the application domain in terms of their properties and their relationships with other objects (inter/intra object relationships) with respect to time. Temporal operational properties describe the behaviour of objects over time, as reflected through changes in their properties. Finally, temporal constraint properties describe conditions the object properties must satisfy during the object lifespan. More specifically,

1. A *temporal object* is defined as a set of one or more temporal properties. The temporal properties describe structural, operational and constraint characteristics of objects over time.

2. A *temporal constraint* or rule is a database rule that includes also its validity period and is divided into three parts namely *event*, *condition* and *action* part. All these parts may refer to time points but at least the event or condition part do so in order for the rule to be characterized as a temporal rule.

Based on these definitions, and after many hours of discussions the following consensus were achieved.

- The design of the temporal query language associated with the above concepts should take into consideration the traditional language design issues such as ease of use, optimizability, expressiveness and implementability.

- Schema evolution should be supported in a way that will accommodate object persistence across schema changes. The issue of dynamic schema evolution is very important in object-oriented databases and time support can provide approaches to deal with this issue.

- The participants agreed that the modeling of time should be independent of the particular choice for the data model. This means that irrespective of the data modeling concepts, time has an ontology by itself that needs to be defined and agreed upon.

- Besides the basic infrastructure concepts of the next generation of temporal data models, it is possible to define additional constructs for the declaration of conditions/constraints (e.g., homogeneity) which may be beneficial in application development.

The next generation of temporal databases should explicitly support a rich set of time concepts. The workshop participants identified the following concepts as the minimum set of concepts to be incorporated: bitemporal interval, bitemporal span, bitemporal chronon, bitemporal element, bitemporal time point, and operations on intervals, spans and time points as well as conversion facilities between them.

Although all the above concepts are necessary for a comprehensive treatment of time in databases, the participants singled out the notions of interval, span and time point as the key concepts upon which the other concepts can be defined.

4.4 Active and Temporal Database Concepts

To reach consensus on the subject of Active Temporal Databases, there first has to be a shared understanding of the structure of rules and the definition of event. Accordingly, we first provide consensus definitions of active rules and events (in the context of rule definition). We limit the discussion to active rules (ignoring, for instance, deductive rules).

An active rule is an Event-Condition-Action (E-C-A) rule, where

Event E: is a *basic* or *composite* event, as defined below.

Condition C: is either (1) a boolean expression, or (2) a query in the database query language that results in a TRUE/FALSE answer. The query must be side-effect free.

Action A: is an execution of a database operation or an arbitrary application program.

Definition 4.1 Basic Event: a pair *(event occurrence, time instant)*. The event occurrence is represented by some symbol e and is mapped to a time instant t on the system clock. The basic event (e, t) is said to occur at time t. □

It should be noted that an event occurs at a point in time. Examples of basic events include `begin transaction`, `after commit`, and `before read`. In fact, basic events can be obtained from most database operations by adding the modifiers `before` or `after` to the name of the database operation. External signals, and time events, such as *11:00am*, are permitted as basic events. A time event is represented by the pair *(time name, time instant)*, where time name is the symbol representing the event occurrence. `(''11:00am'', 11:00am on July 20, 1993)` is an example of a basic event.

Events may have attributes. For instance, an `after insert` event has information regarding the specific relation updated as well as the specific tuple inserted. Such information is an attribute of the event. In addition, event attributes may include system-level information such as transaction id, user id, and time.

Definition 4.2 Composite Event: can be created from basic and other composite events through the use of a closed algebra. A composite event occurs at a time instant, as specified by the closed algebra in terms of the time instants of component basic events. □

Several algebras for composite events have been proposed, e.g., ODE [GJS92, GJMS93]). We permit simple conditions, such as X > 10< on attribute X of an event, or a boolean predicate on attributes of events, to be included in the algebra for composite events. Note that the boolean predicates in a composite event do not refer to items stored in the database, and can be evaluated from the given event, without querying the database. Permitting boolean expressions allows for easy specification and efficient implementation of events such as

every n^{th} trade of IBM stock at price > 50

that would otherwise require complex temporal queries in the *Condition* part. A more complex algebra may also permit predicates that refer to database items, We assume that an optimizer that can move such complex predicates into the condition part would be provided, thereby making the algebra equivalent to the one we consider.

Both basic and composite events are usually referred to by event names or event identifiers. Depending upon the event description, and the type of database, a rule may get associated with one or more relations, views, or objects. Rules can typically be inserted, deleted, updated, activated, and deactivated by the user as well as by the system.

To summarize, for the purpose of this report, we will assume that the event part of a rule is based on an algebra, that the algebra will permit certain conditions to be included in the event part, and that there will be a separate *Condition* part in the rule. For high-level syntax, we will express a rule as "WHEN event IF condition THEN action"; different syntax may be used and defaults assumed in actual implementations. It is desirable of course to standardize on a rule language.

4.5 Temporal Rules

Following the above definitions of rules and events, we focus on the definition of temporal rules, and distinguish between two cases: temporal rules in non-temporal databases and temporal rules in temporal databases.

Definition 4.3 Temporal Active Rule: An active rule is said to be temporal if (1) the event is a composite event that refers to basic events occurring at time points other than the time when the rule is fired, or (2) the event refers to explicit time basic events, or (3) the condition contains a temporal database query that cannot be expressed in a non-temporal query language that can reference the basic event (or the last basic event in the composite event that caused the rule to fire), operating over a database that does not maintain a temporal history. □

Note that the action is not mentioned in the definition of a temporal active rule. The action is an arbitrary procedure, and we will not attempt to characterize a rule based on the behaviour of the action.

In the above definition of a temporal active rule, conditions (1) and (2) may be seen as the definitions of a *temporal event*, and condition (3) as the definition of a *temporal condition*.

4.5.1 Temporal Rules in a Non-temporal Database

In a non-temporal database, the query language is non-temporal, so the condition of a rule cannot contain a temporal query. Hence, active rules in a non-temporal database are temporal if and only if (1) the event is a composite event that refers to basic events occurring at time points other than the time when the rule is fired, or (2) the event refers to explicit time basic events.

We consider both these cases:

Composite events ⇒ Temporal Rules Composite event algebras enable one to relate basic events occurring at different points in time. One can specify simple patterns of such events that are of interest, in much the same way as a temporal query can specify patterns of values in successive versions of relations. Composite events thus represent simple forms of temporal queries, and can provide simple temporal features. Active rules using such event algebras must therefore be considered to be *temporal rules*. The composite algebras can be used in non-temporal databases, provided mechanisms to recognize these event patterns are provided, e.g., [GJS92]. While we will not discuss any particular algebra in this paper, we will illustrate their relationship to temporal databases through a representative syntax.

EXAMPLE 4.1 Consider an inventory database in a store. There is an inventory(item, amount) relation storing the amount of each item in stock. Further, at the end of every month, sales statistics for the month are computed. One of the statistics is the average price and quantity sold for each item in the store during the last month.

We want to label an item as ***high-tech*** if it sold in low quantities and at high prices for three consecutive months some time in the past, and has been selling in high quantities and at low prices for the last three consecutive months. Clearly, in a temporal database, a temporal query can be used to identify the ***high-tech*** items. We show how a composite algebra can be used to define a temporal rule that labels items as ***high-tech***.

Let *u* be an item carried by the store, and let u_sale(Q, P) be an event representing the insertion of the monthly statistic "item *u* sold in quantity Q at average price P during the last month".

We first define derived events u_losale and u_hisale that represent the facts that (1) item *u* sold in low quantities at high prices, and (2) item *u* sold in high quantities at low prices, respectively, in the last month. Assume LO_QTY, LO_PRICE, HI_QTY, and HI_PRICE are system constants defined elsewhere.

```
(r1 ):  #define u_losale = u_sale(Q, P) && Q < LO_QTY
            &&  P > HI_PRICE;
        #define u_hisale = u_sale(Q, P) && Q > HI_QTY
            && P < LO_PRICE;;
```

Suppose that, when we identify an item to be *high-tech*, we want to check if its current stock is greater than HI_QTY, and if not, we want to place an order for an amount =

HI_QTY less the currently stocked quantity of the item. One may chose to write this as follows (the syntax below is for purpose of illustration only, we are not promoting this syntax. For instance, in a real language, one would have higher order constructs to represent repetition).

```
(r2 ):  WHEN sequence(u_losale, u_losale, u_losale)
             followedby
             sequence(u_hisale, u_hisale, u_hisale)
        IF (SELECT amount
            FROM inventory
            WHERE item = 'u' AND amount < HI_QTY)
        THEN order('u', (HI_QTY - amount));
```

The WHEN part of the active rule uses two composite algebra operators: **sequence** and **followedby**. **sequence(u_losale, u_losale, u_losale)** is a composite event that occurs when a sequence of three consecutive **u_losale** events occur, at the point when the last **u_losale** event in the sequence occurs. The composite event (a **followedby** b) occurs if the event b occurs sometime after event a has occurred, at the point in time when event b occurs. So, the composite event in the WHEN clause occurs at the point in time when the third month's high sale figure is reported, and at some time in the past, three consecutive low sale figures were posted. The IF part of the rule is a condition that checks if the given SQL query returns a nonempty answer. In case it does, an order is placed for the difference between **HI_QTY** and the **amount** returned by the SQL query. □

The active rule $r2$ is considered to be a temporal rule because it can be mapped to an equivalent rule where the WHEN clause contains basic events, and the IF clause contains a temporal query, providing such a query was permissible in the system. For example, if the WHEN part was limited to basic events, then it would contain the event **u_hisale**, and the IF part would need a temporal query that refers to old versions of a sales relation. We assume here that the sales relation only keeps the average price and total quantities for the last one month. Since an active rule that has a temporal query in the condition part and a basic event in the event part would definitely be called a temporal rule, we must also consider the equivalent rules of type $r2$ as temporal.

One may want to enhance Example 4.1 to:

1. Require that the sequence of **u_losale** events be followed by the sequence of **u_hisale** events within one year.
2. Check whether the current inventory amount is less than the sum of the last two months sales, and if so, to order the difference between the sum of the last two months sales and the current amount in the inventory.
3. Rather than writing a separate rule for each item that one wants to track in a similar fashion, write one active rule to track all such items.

These enhancements require that events have attributes, and that attributes of events be passed across time, to other events, and into the condition and action part [GJMS93].

Explicit Time Events ⇒ Temporal Rules The event can contain explicit reference to times, such as at "11:00am on Jan 26, 1950". A *calendar algebra* can be defined to refer to time events at a higher level, such as "3rd Friday of a month". In either of the above cases, we say that the event refers to basic time events, so the active rule is temporal. Such basic time events can be used like any other basic events in defining composite events using a composite algebra.

An example of a temporal rule using basic time events is, "**WHEN every 10 Minutes IF** condition **THEN Evaluate(Portfolio)**", where **Evaluate** is a user-defined procedure that is applied to the named object (condition may be any condition on the database states).

4.5.2 Temporal Rules in Temporal Databases

From the definition of temporal rules, it follows that an active rule in a temporal database is nontemporal if the event is basic and the condition contains a non-temporal query that could be expressed in a non-temporal query language. All other rules in a temporal database are called *temporal rules*. Thus, rules that would be considered temporal in a non-temporal database, are also considered temporal in a temporal database.

Further, both temporal and non-temporal active rules can be be viewed as first class database objects. This means that the history of rules should be kept. Each rule is associated with transaction and valid times. Transaction time is the time when the rule was recorded in the database. Valid time represent the time point(s) when the rule is applicable, i.e., checked for activation. The valid time of a rule can be specified explicitly as a temporal element, or implicitly in terms of data condition(s) or the occurrence of some event(s). Activation and deactivation of rules is achieved by changing their valid time.

There are two basic alternatives for modeling the history of rules. In the first way, the rule is considered as a single unit, and thus, a change to one of the components is regarded as deletion (note: in the temporal database case, deletion of a rule amounts to indefinite deactivation) of the rule and the addition of a new rule. In the second way, a rule is considered to be a complex object and the history of the individual components is maintained, that is, we can represent different versions of the same rule.

4.5.3 Actions of Rules in Temporal Databases

When a rule is activated, and the condition evaluates to true, the action part of the rule gets executed. In a temporal database, the action can include any update to the database, including updates to past or future valid times of data items. Such updates are called retro-active and pro-active updates, respectively.

If proactive or retroactive updates are allowed in a temporal database, rules can effect data in several ways. We will elaborate on the retroactive case only; dealing with proactive updates is similar. In [EGS93a] the following characterization of retroactive effects is given In the following definitions, "past" is measured relative to the time of the operation or triggering of a rule:

Definition 4.4 Retroactive Update: an update operation that modifies past values of data elements. □

Definition 4.5 A Retroactive Rule: a rule whose action includes a retroactive update. □

Definition 4.6 Retroactive Rule Activation: the application of a rule to past snapshots. □

These definitions indicate that rules can effect data *retroactively* in two main ways: due to retroactive rules, or due to retroactive activation of rules. The latter case can occur for two reasons: 1) a rule is introduced in the system with a valid time that includes past time interval(s), or 2) a retroactive update occurred, and the updated data element(s) trigger a rule which was valid at that past time.

4.6 Temporal Consistency

Generally speaking, the consistency of a database is measured relative to the effect of a serial execution of a set of transactions on a state that is assumed to be consistent (i.e., the serializability condition), and relative to a set of constraints that limit the space of legal database states. In the rest of this section we assume that the serializability condition is satisfied, and therefore, consistency is in the context of constraints only. Constraints can be non-temporal, i.e., they refer to any valid time snapshot, or temporal, i.e., they refer to particular snapshot(s). It is assumed that constraints can be compiled into rules that enforce them. Note that these rules can also derive data items.

In order to characterize the actions of rules in a temporal database, there is a need to distinguish between a database state and a snapshot. The following definitions refer to a *bitemporal* database, i.e., a database that supports both transaction and valid times. We use the term ***system time*** to refer to the time values generated by the system clock. It is assumed that these time values are used as the domain of transaction time. *Observation time* refers to the reference point in the system time line from which the database state is observed. In conventional databases the observation point is always NOW. In temporal databases the observation point can be less than or equal to NOW. Only data objects with transaction time less than the observation time can be seen by a query (or a transaction).

We define a few concepts needed to understand temporal consistency.

Definition 4.7 Database State(t): all the values of data objects committed by system time t. □

Since we assume no overwriting of data, each state contains the complete database evolution up to time t. Moreover, the history of database states is kept as well, and therefore is a history of histories (or a sequence of sequences). Note that database states are ordered by system time.

Definition 4.8 Transaction Time Database Snapshot(t): the database state at system time t. It is assumed that this snapshot is the same for any observation time greater than t. □

Note that Transaction Time Database Snapshot is the same as the database state at the specified time. The reason the two definitions are given is that those two terms are used by many people with different meaning. It is convenient in some contexts to use the term database state and in others to use transaction time snapshot. However, these are synonyms.

Definition 4.9 Valid Time Database Snapshot(t_1, t_2): the world's state (as inferred from the database states) at valid time t_1 as observed from system time t_2. t_1 can be either greater than or less than or equal to t_2 (greater than implies that the data values are predictions). □

At the presence of retroactive or proactive updates, a snapshot characterization requires the specification of an observation point, i.e., the snapshot values can be different for different observation points. Thus the value of a data object at a given valid time is a function of the observation time. Consequently, the consistency of the database has to be determined relative to a chosen observation time.

Definition 4.10 Temporal Consistency at Time t: An active temporal database is consistent at system time t if for all valid time instants t_v, a valid time snapshot at time t_v as seen from time t (which will include all the data objects whose valid time intervals include t_v when observed from time t) satisfies all the rules that are valid at time t_v. □

4.7 Real Time Constraints

The ECA model allows one to capture the condition corresponding to the lack of completion by a deadline but not much more. While active databases possess the necessary features to deal with many aspects of real-time database systems, the crucial missing ingredient is the active pursuit of the timely processing of actions.

4.8 Conclusion and Follow-on Efforts

The overall objective of the discussions in group C was to identify the common infrastructure for the next generation of temporal database concepts including extensions of the relational data models as well as the adoption of concepts from object based data models. The emphasis of the discussions was on object based models since the participants felt that this is the most likely way forward. Research work in the areas of temporal object bases and temporal active databases is quite preliminary and consequently, the infrastructure is less well developed here as compared with that for temporal relational databases.

The participants of group C1 discussed in some detail the future directions of the work in this area. Although, it is too early to consolidate, the participants felt that any future work on infrastructure should be linked with work on SQL3. The main reason for that is that SQL3 is still open for negotiation and in addition, incorporation of time semantics into it will certainly have a major impact in the community unlike the work in temporal relational databases and extensions of SQL-89 and SQL-92.

Future work in this area should also be linked with the findings and conclusions reported in the next section, in order to provide the required performance levels necessary for the wide acceptability of temporal object database technology. This link was not investigated in any detail during this workshop but it will certainly be a major issue for discussion at a future infrastructure workshop.

Finally, work on the glossary should incorporate concepts from any proposed extensions to SQL3 and agreed infrastructure for temporal object databases.

5 Group D: Implementation

5.1 Introduction and Group Charter

Working group D, consisting of J. Blakeley, R. Elmasri, S. Jajodia, V. Kouramajian, K. Makki, D. Peuquet, V. Tsotras, and D. Wells, was concerned with the definition of system implementation techniques and an architecture for temporal databases. The identification of any similarities and differences between an architecture for a temporal DBMS and that for a non-temporal DBMS was one of our goals. Proposing a reference architecture was one of the goals of the group.

A second goal was to identify a suitable model, or models, for representing temporal databases at the storage level. Such a model would be needed for discussions on several system modules, such as query optimization, it was argued. As it turned out, our discussions led us to change this opinion.

A third goal was to identify which aspects of a temporal database, if any, need to be identifiable at the storage level; for example, whether such time dimensions as valid and transaction time would need to be explicitly represented at the storage level.

The fourth goal was to identify a number of system modules (query optimization, concurrency control, security, etc.) and to determine preliminary requirements for each of these modules.

Finally, we wanted to discuss what are typically temporal database applications in preparation for the specification of performance benchmarks. These benchmarks would be used to compare proposed indexing and storage structures for temporal databases.

5.2 Current Status of the Field

Only a few generalized temporal database management systems have been implemented. The TQuel prototype [AS86] is perhaps the best known. However, because many applications of databases are inherently temporal, there have been countless implementations of ad-hoc temporal databases that either utilize existing commercial DBMSs or that build temporal databases over file systems. In these implementations, the meaning and interpretation of time is implemented by the user application programs rather than being understood by the DBMS software itself, which is the goal of a generalized temporal DBMS.

A number of indexing techniques have been proposed that claim to improve the performance of search based on temporal conditions. Some are extensions of techniques that were originally proposed for spatial indexing, whereas others were explicitly designed for temporal databases.

A crucial aspect of a temporal DBMS architecture is to define a standard algebra that is well accepted for temporal database operations. This would correspond to the well-accepted relational algebra operations for representing non-temporal database requests. In addition, a set of update operations for temporal databases would be useful. These operations would be the target internal representation for temporal queries and updates, and would serve as a basis for such system modules as query processing and optimization. Unfortunately, there is no well-accepted temporal database algebra (there is, however, no shortage of candidates [MS91a]).

There has been little research in areas such as identifying concurrency control, recovery, security, and other techniques that would take advantage of temporal database features, such as the availability of the history of database changes.

5.3 Baseline Architecture

After heated discussion, there was general agreement that the architecture of a temporal DBMS should not differ drastically from that for a non-temporal DBMS. In particular, it seems that at the physical storage level (i.e. disk pages), data can be stored as byte streams as for non-temporal databases. However, we did not have enough time to discuss the impact of time on concurrency control, recovery, and security mechanisms. We mainly were considering the system modules for query processing and optimization.

Our baseline architecture consists of four main modules. At the lowest level, a storage system exists, which stores persistent data in disk pages. This level could use existing storage systems, such as EXODUS or the UNIX file system. Stored objects are retrieved as byte streams, and interpreted as objects at a higher level of the system based on the information stored in the system catalog. We could not agree whether the basic storage model should be based on tuple versioning, attribute versioning, the use of deltas, or a combination of these techniques. At the storage level, data records can be clustered for efficient access. For example, the partitioning of storage into a current store and history store could be used. Indexes to locate related data or to search based on time conditions, attribute values, or a combination of both, could be built.

Above the storage level, there would be a number of higher level modules. One module would include an extensible library of available execution algorithms is proposed. Another module that contains a library of available index structures would be accessible by some of the execution algorithms. The execution algorithms would be various implementations of the high-level temporal database operations. A query optimization module would create an execution strategy for a temporal query by choosing the appropriate options from the execution algorithms library. With reference to the conceptual architecture shown in Figure 1, these two lower levels (storage system, query optimization and evaluation) correspond to the representational side of that figure.

Standard modules such as query parser would create an internal query representation, which would then be optimized by the query optimizer (query parsing and logical query optimization corresponds to the right middle of Figure 1, concerned with the (single) conceptual data model on which the temporal query language is based.

The issue of which set of formal operations to use in representing and optimizing temporal queries was not resolved. There were two main points of view. The first was that we should extend the standard relational algebra operations with the interval algebra [All83] so that we can proceed with prototype implementations and analysis of various optimization methods, indexing techniques, and execution algorithms. The second point of view was that we do not fully understand how temporal databases differ from non-temporal ones, and that we should examine new algebras developed explicitly for temporal databases. The conclusion was that we should proceed in both directions, with some researchers taking the first shorter-term approach, while others pursue a possible long-term better solution.

The indexing module should be extensible. Thus, new indexing methods would be added to the library as they become implemented. For each indexing method, the specification of the storage model that it is compatible with, as well as the execution modules that can use it, and cost estimate functions for use by the optimizer, must be given when it is added to the index library.

5.4 Performance Benchmarks

Performance benchmarks to compare various proposals for temporal index structures and search techniques are needed. We agreed that there is probably no typical temporal database application, so that it would be necessary to create a number of benchmarks for different applications. The following characteristics should be considered when designing a temporal benchmark: database size, frequency of updates, archiving characteristics, presence of retrospective updates, and query characteristics. The metrics to be measured by a benchmark include the space consumption by indexing and storage structures, the update time, the archiving/migration time, and access times for different types of queries.

5.5 Extensible Query Processing Architecture

We further discussed the query processing architecture for temporal databases and agreed that it should support extensibility at various levels. At the algebra level, the architecture should support the use of different algebras; for example, a relational algebra extended with Allen's interval algebra, or one of the many algebras proposed by temporal database researchers. The set of execution algorithms in the execution algorithms library should also be extensible. New search techniques can be incorporated by adding their implementations and descriptions to the library, along with cost estimation formulas to be used by the optimizer. The optimization algorithms themselves may be changed by basing them on different paradigms, such as dynamic programming or branch and bound.

This organization is consistent with the trend towards open architectures.

5.6 Conclusion and Follow-on Efforts

In summary, our group recommends that the architecture for temporal databases be based on similar architectures for non-temporal databases. The emphasis is on flexibility so that various query optimizers, execution algorithms, index techniques, and storage models could be supported. We recommend two levels of future investigation: short term and long term.

For the short term, we should examine in more detail the suggested framework for temporal query processing and optimization. The proposed system modules and their interactions should be further specified. We recommend that an algebra for internal representation of temporal queries be developed based on extending the existing relational algebra with temporal operations. Research to optimize temporal queries based on this algebra would then proceed. A library of execution algorithms and a library of indexing methods that can be used to implement the operations of this algebra would be needed. The characterization of a number of benchmarks for various temporal database applications is needed to be able to compare various optimization techniques and indexing methods.

For the longer term, we recommend that research continue in identifying whether or not there are more significant differences between temporal and non-temporal databases. The use of proposed temporal algebras that are more independent from the relational algebra as a basis for system implementation should be investigated. The impact of temporal databases on concurrency control, recovery, security, and other system modules should be investigated.

6 Conclusions

This workshop was the first opportunity for those active in temporal databases to meet and discuss the common aspects of extant ideas and proposals. The workshop was unusual in that the topics of discussion were not new results of research, nor recommendations for future research, but rather which results of previous research could be identified as common infrastructure.

The preceding four sections each enumerate specific contributions to an infrastructure for temporal databases. There were several common threads that ran through many of these individual group discussions.

- *The infrastructure must be based on a core set of desired features, so that most temporal applications receive at least some support from the temporal DBMS.*

  Applications demand a wide variety of temporal database features, from storage of timestamps through temporal joins through support for relative time (in which only the relative ordering of events is known), through full-fledged temporal reasoning. Because of this diversity of requirements, the infrastructure should only include those aspects that support a significant fraction of the applications, and that are fairly well understood.

- *Terminology is critical.*

  As time is such a prevalent aspect of data, and indeed of life in general, it is natural that different spheres of activity would come up with different terms for the same concept (e.g., an airplane trip from New York to Paris is a "macro-event" to some and an "interval" to others) and identical terms for different concepts (e.g., an "event" to some is simply a position on a time line, whereas to others it is an occurrence of something interesting). Much effort was invested to develop a well-defined glossary of relevant terms.

- *Aspects of the conceptual model must be separated from concerns of the representation.*

  This separation proved to be beneficial in several of the discussions: it enabled the issue of performance to be separated from the issue of semantic integrity. Particularly in databases, performance is seen as all-important, with other issues subjugated to a lesser status. Several of the groups made explicit distinction between the semantics of the data, as expressed in the conceptual model, from the encoding of the data, as expressed in a representational model.

- *The baseline architecture must be extensible, and must identify what is different about a temporal DBMS, and what can vary between TDBMS implementations.*

  The distinction of conceptual versus representational is incorporated into the architecture. Extensibility of the storage model and index library ensures that different representational models can be employed, thereby achieving high performance through the use of storage models and temporal indexes appropriate to the application.

In addition to this report, several other components of an infrastructure for temporal databases have recently been completed.

Substantial effort over the two years preceding the workshop generated an initial glossary that was published in the *SIGMOD Record* [JCG+92], and its impact on standardizing terminology is now being felt. Christian S. Jensen headed an editorial board to complete the glossary. The glossary, containing 87 terms and their definitions, appears in this issue.

Also, over the six months prior to the workshop a fairly exhausting consensus effort generated an initial draft of a "language benchmark" intended to be an aid in evaluating the user-friendliness of proposals for temporal query languages. Christian S. Jensen spearheaded the effort to complete "test suite of temporal query languages", as it is now called. This document is focused on SQL language extensions.

Several other efforts contemporaneous with the planning of the workshop also contribute to an infrastructure for temporal databases. The first book on temporal databases [TCG+93] is a comprehensive volume covering modeling, languages, and implementation aspects of temporal databases. The book consists of 23 chapters that report the research results of leading researchers in temporal databases. The fifth in a series of bibliographies on temporal databases appeared in the December, 1993 issue of *SIGMOD Record*, a bibliography on spatiotemporal databases appeared in the March, 1993 issue of *SIGMOD Record*, and an extended version will appear in the *International Journal of Geographical Information Systems.*

Several consensus efforts were started as a result of the discussions at the workshop. The glossary is continuing, and new terms will be added as temporal databases and their diverse applications are better understood. The TSQL2 and TSQL3 language design efforts are ongoing. In particular, the TSQL2 language design committee has released an initial language specification, in this issue.

7 Acknowledgements

Curtis Dyreson contributed to the preparation of Section 2. Fabio Grandi's notes were helpful in the preparation of Section 3.

This report benefited from the viewpoints expressed by the workshop participants, listed in Table 1.

This workshop was supported by ARPA and the National Science Foundation under grant IRI-9304091. Support was also provided by Bell Communications Research and Tandem Corporation. However, the findings expressed in this report do not necessarily reflect the official position of any of the sponsoring agencies.

References

[All83] J.F. Allen. Maintaining Knowledge about Temporal Intervals. CACM, 26(11):832–843, November 1983.

[Ariav93] G. Ariav. Tools for managing temporally oriented data: are they really prectically relevant? In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[AS86] I. Ahn and R. Snodgrass. Performance Evaluation of a Temporal Database Management System. In *Proceedings of the SIGMOD Interna-*

tional Conference, Washington, DC, pp. 96–107, May 1986.

[BB93] A.P. Buchmann, H. Branding. On Combining Temporal and Real-Time Databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[CCT93] J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical relational query languages. Technical report STERN IS-93-8, New York University Stern School of Business, 1993. (to appear in TODS).

[Cha92] S. Chakravarthy. Architectures and monitoring techniques for active databases: An evaluation. UF-CIS TR-92-041. (Submitted to Applied Data and Knowledge Engineering Journal).

[Chen76] P.P.-C. Chen. The Entity-Relationship Model-Toward a Unified View of Data. ACM TODS 1(1):9–36, March 1976.

[Date88] C.J. Date. A proposal for adding date and time support to sql. *ACM SIGMOD Record*, 17(2):53–76, June 1988.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ladin, D.R. McCarthy, A. Rosenthal, S. Sarin, M.J. Carey, M. Livny, and R. Jauhari. The HIPAC project: Combining active databases and timing constraints. *ACM SIGMOD Record*, 17(1):51–70, March 1988.

[DW90] C. J. Date and C. J. White. *A Guide to DB2*, Volume 1, Third edition. Addison-Wesley, Reading, MA, September 1990.

[EGS93a] O. Etzion, A. Gal, and A. Segev. Retroactive and Proactive Database Processing. Technical Report LBL-34424, Lawrence Berkeley Laboratory, July 1993.

[GJS92] N. Gehani, H.V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In *Proceedings of the Eighteenth International Conference on Very Large Databases*, pp. 327–338, Vancouver, Canada, August 1992.

[GJMS93] N. Gehani, H.V. Jagadish, I.S. Mumick, and O. Shmueli. Temporal Queries for Active Database Support. In *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[JCG+92] C.S. Jensen, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass. A glossary of temporal database concepts. ACM SIGMOD Record, 21(3):35–43, September 1992.

[Kline93] N. Kline. An Update of the Temporal Database Bibliography. SIGMOD Record, 22(4):66–80, December, 1993.

[MD89] D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, pp. 215–224, Portland, OR, May 1989.

[MS93] J. Melton and A.R. Simon. Understanding the New SQL: A Complete Guide. Morgan Kaufmann, 1993.

[MS91a] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. ACM Computing Surveys, 23(4):501–543, December 1991.

[OC87] Oracle Computer, Inc. *ORACLE Terminal User's Guide*. Oracle Corporation, 1987.

[Rama93a] K. Ramamritham. Real-Time Databases. Journal of Distributed and Parallel Databases, 1(2):199–226, 1993.

[SA86] R.T. Snodgrass and I. Ahn. Temporal databases. IEEE Computer, 19(9):35–42, September 1986.

[SS92] M. Soo and R. Snodgrass. Mixed Calendar Query Language Support for Temporal Constants. TempIS Technical Report 29, Computer Science Department, University of Arizona, Tucson, Arizona, Revised May 1992.

[TC83] Tandem Computers. *ENFORM Reference Manual*. Cupertino, CA, 1983.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.). *Temporal Databases: Theory, Design, and Implementation*. Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, 1993.

[WF90] J. Widom and S.J. Finkelstein. Set-oriented production rules in a relational database system. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pp. 259–270, Atlantic City, NJ, May 1990.

Participant	*Group*	*Affiliation*
Ilsoo Ahn	B	AT&T Bell Laboratories
Gad Ariav	A	Tel Aviv University, Israel
Marianne Baudinet	A	Universite Libre de Bruxelles, Belgium
José Blakeley	D	Texas Instruments
Mark Boddy	A	Honeywell Systems and Research Center
Alex Buchmann	C.2	Technisch Hochschule Darmstadt, Germany
Sharma Chakravarthy	C.2	University of Florida
Su-Shing Chen		National Science Foundation
Tsz-Shing Cheng	C.1's rapporteur	Iowa State University
James Clifford	B	New York University
Klaus Dittrich	C.2	Universitat Zurich, Switzerland
Curtis Dyreson	A's rapporteur	University of Arizona
Ramez Elmasri	D's coordinator	University of Texas at Arlington
Max Egenhoffer	A	University of Maine
Shashi K. Gadia	C.1; visited B	Iowa State University
Fabio Grandi	B's rapporteur	Università di Bologna, Italy
Pat Hayes	A	Beckman Institute
Sushil Jajodia	D	George Mason University
Christian S. Jensen	B	Aalborg Universitetscenter, Denmark
Wolfgang Käfer	B	Universitaet Kaiserslautern, Germany
Vram Kouramajian	D's rapporteur	University of Texas at Arlington
Krishna Kulkarni	B	Tandem Corporation
Ted Lawson	C.1	University of Wales College of Cardiff, U.K.
Nikos Lorentzos	B	Agricultural University of Athens, Greece
Kia Makki	D	University of Nevada
Inderpal Singh Mumick	C.2	AT&T Bell Laboratories
Frank Olken	A	Lawrence Berkeley Lab
M. Tamer Öszu	C.1	University of Alberta, Canada
Barbara Pernici	A's coordinator	Politecnico di Milano, Italy
Donna Peuquet	D	Pennsylvania State University
Niki Pissinou	C.1	University of Southwestern Louisiana
Krithi Ramamritham	C.2	University of Massachusetts
Scott Relan		Digital Systems Research
Arie Segev	C.2's coordinator	University of California at Berkeley
Richard Snodgrass	B's coordinator	University of Arizona
Michael Soo	C.1's rapporteur	University of Arizona
Sury Sripada	A	ECRC, Munich, Germany
Stanley Su	C.1	University of Florida
Abdullah Uz Tansel	B	City University of New York
Babis Theodoulidis	C.1's coordinator	UMIST, Manchester, U.K.
Vassilis Tsotras	D	Polytechnic University
David Wells	D	Texas Instruments
Gio Wiederhold		ARPA/SISTO
Gene Wuu	C.1	Bell Communications Research

Table 1: Workshop Participants