

An Efficient Method for Indexing Now-relative Bitemporal data

Bela Stantic

Sankalp Khanna

John Thornton

Institute for Integrated and Intelligent Systems
Griffith University, Brisbane Australia,
PMB 50 Gold Coast Mail Centre Queensland 9726 Australia,
Email: {B.Stantic,S.Khanna,J.Thornton}@griffith.edu.au

Abstract

Most modern database applications involve a significant amount of time dependent data and a substantial proportion of this data is *now-relative*, current *now*. While a lot of work has focussed on indexing on temporal data in general, only a little work has addressed the indexing of *now-relative* data, which is a natural and meaningful part of every temporal database as well as being the focus of most queries. This paper proposes a logical query transformation that relies on the *POINT* representation of current time and the geometry features of spatial access methods. Logical query transformation enables off-the-shelf Spatial indexes to be used. We empirically prove that this method is very efficient on *now-relative* Bitemporal data, outperforming the straightforward maximum-timestamp approach by over a factor of 20, both in number of Disk accesses and CPU usage.

Keywords: Bitemporal Databases, Access methods, now-relative data, performance

1 Introduction

Research in the field of Temporal Databases has sparked very keen interest over the last two decades with a lot of papers submitted by several hundred researchers (Date, Darwen & Lorentzos 2002), (Jensen 2000), (Snodgrass 2000). Despite being a crucial element in the criteria governing their application, access methods for temporal data picked up momentum as a research topic much later and most of the proposed access structures are based on the family of spatial R-trees indexing techniques.

In temporal databases time can be captured along two distinct time lines: transaction time and valid time (Jensen & Snodgrass 1999). The valid time line represents when a fact is valid in the modelled reality and the transaction time line represents when a transaction was performed. A Bitemporal database is a combination of Valid time and Transaction time databases and records the database states with respect to both valid and transaction time, which are considered to be orthogonal (Snodgrass & Ahn 1986). By supporting both valid time and transaction time lines, it represent reality more accurately than conventional databases. However, being append only, they are usually very large in size, and the informational benefits of Bitemporal data can, without effective management, be easily outweighed by the costs of poor access times. The need for efficient access methods is thus more crucial in Bitemporal Databases than

in conventional databases.

A lot of multidimensional access structures have been proposed and some of them have been recommended for temporal databases. Despite *now-relative* data, which are facts that are valid *now* and/or tuples that belong to the current database state, being a natural part of Bitemporal databases, only a few access methods have addressed this issue.

To represent that a fact is valid *now* or that a tuple belong to the current database state a special variables are used to represent current time, such as “now”, “UC” (until changed) or “∞”. Problems with having variables that represent current time in index structures have been identified in the literature and several index structures that avoid variables have been proposed. The GR-Tree and the 4R-Tree support both *now-relative* valid and transaction time without using variables (Bliujute, Jensen, Saltenis & Slivinskas 2000). Essentially both seek to improve the efficiency of the underlying structures by reducing dead space in the tree formed by using the maximum-timestamp approach in a standard R*-Tree. Previously proposed techniques require query transformations and modification to the kernel, so off-the-shelf R-tree spatial indexes can not be simply reused.

In the reminder of this paper we will refer to the *MAX* approach when current time is represented with maximum-timestamp that Database Management System supports. While for approach where current facts or current tuples are represented as points, with end times equal to start times, we will refer to it as the *POINT* approach (Stantic, Thornton & Sattar 2003).

In this paper we present a method to index *now-relative* Bitemporal data by logically dividing the regions and additionally employing the geometry types feature of spatial indexes. We perform query transformations without requiring modification to the kernel, so we use off-the-shelf Spatial R-tree index.

Because our approach reduces the dimensions of the Spatial geometries, typical rectangles reduces to lines or points, this impacts positively as the geometries will take up less room in the leaf nodes leading to higher fanout and better performance. The smaller dimensions of the *now-relative* data mean that there is less overlap between the maximum bounding regions, which form the boundaries of each node. In addition, because the representation of *now-relative* data generally do not extend beyond current time the amount of dead space is reduced (nodes that are searched that do not contribute to the answer).

Further, we give the first empirical proof in real environment, of the earlier hypothesis that the straightforward maximum-timestamp approach is inefficient in the handling of *now-relative* Bitemporal data (Bliujute, Jensen, Saltenis & Slivinskas 1998). The *POINT* approach, however proved to be extremely efficient.

In the remainder of the paper, we first look more closely at temporal data concepts. In section 3 we briefly introduce existing methods to index *now-relative* Bitemporal data and highlight their limitations and disadvantages. In section 4, we present the “core” of the *POINT* approach by presenting the extension to the query notation and the logical query transformations. Section 5, explains the experiment taken to evaluate the *POINT* and *MAX* approach using a real database environment, further it defines the measurables used to evaluate results with respect to Disk reads and CPU usage, and an analysis of the results. Finally, in section 6, we present our conclusions and discuss possible extensions and future work.

2 Temporal Data Concepts

A temporal database is one that supports some aspect of time distinct of user-defined time. While being ever changing, time is an important aspect of all real world phenomena. Each event bears a time attached to it, sometimes in more than one form. Time marks the starting and ending of an event and establishes the validity of data. Facts, i.e. data valid today may have had no meaning in the past and may hold no identity in the future. Some data, on the other hand may hold a historical significance or may continue to be valid up to a predefined period in time. This relationship of time and data adds a temporal identity to most data and in this light it would be hard to identify applications that do not require or would not benefit from database support for time-varying data. Most current database systems represent a single state of data and this is most commonly assumed to be the current state. Any modifications normally result in the overwriting of the data and the old data being discarded. Though commercial databases offer capabilities to keep track of the old (Oracle 9i Flash-Back) , discarded data, it is solely for the purpose of database recovery and not to retain the old state of the data. Currently, support for time in Conventional Database Systems is almost entirely at the level of user-defined time.

In the literature two time lines of interest have been mentioned, transaction time and valid time. The valid time line represents when a fact is valid in modelled reality (i.e. when it was *believed*) and the transaction time line represents when a transaction was performed. A Bitemporal database is a combination of valid time and transaction time databases where this two time lines are consider to be orthogonal (Snodgrass & Ahn 1986).

The most common notation used to represent Bitemporal data is the TQuel four-timestamp format (Snodgras & et al. 1987) where in addition to non-temporal attributes, each tuple has four temporal attributes - Vt^+ and Vt^- representing the starting and ending time points of the validity of a fact in the modelled world, i.e. the valid time interval, and Tt^+ and Tt^- representing the time when a tuple is inserted in the database and the time it is logically deleted, i.e. the transaction time interval. The time intervals are defined as $[t^+, t^-]$, where t^+ represents the starting time instant and is closed while t^- represents the ending time instant and is open. The valid time interval VT can be represented as:

$$VT = [Vt^+, Vt^-]$$

where start instant is = Vt^+ while end instant is Vt^- and it is not included.

Similarly, the transaction time interval, TT can be represented as:

$$TT = [Tt^+, Tt^-]$$

where start instant is = Tt^+ while end instant is Tt^- and it is not included.

Now-relative data are facts that are valid *now* and/or tuples that belong to the current database state. In the literature to represent that a fact is valid *now* or that a tuple is current, special variables are used, such as “now” , “UC” (until changed) or “∞”. The introduction of variables into temporal databases leads to the under researched area of *Variable databases* (Clifford, Dyreson, Isakowitz, Jensen & Snodgrass 1997). It is our intention to avoid such variables in our approach.

In general the *now-relative* time interval can similarly be defined as $[t^+, CT)$, where the CT represents the current time and is dependent on the approach used. We can now define the domain of Bitemporal data as follows:

Definition 1: The domain of Bitemporal tuples D^{CT} on a domain of Timestamp values T and the domain of tuple identifiers ID in general can be defined as follows:

$$D^{CT} \cong \{ \langle Vt^+, Vt^-, Tt^+, Tt^-, id \rangle \in T \times (T \cup \{CT\}) \times T \times (T \cup \{CT\}) \times ID \mid (Vt^- = CT \vee Vt^+ < Vt^-) \wedge (Tt^- = CT \vee Tt^+ < Tt^-) \}$$

By replacing CT with the maximum timestamp supported by the particular RDBMS, the *MAX* approach to representing current facts or current tuples, it follows:

Definition 2: The domain of Bitemporal tuples D^{MAX} on a domain of Timestamp values T and the domain of tuple identifiers ID for the *MAX* approach for representing current time can be defined as follows:

$$D^{MAX} \cong \{ \langle Vt^+, Vt^-, Tt^+, Tt^-, id \rangle \in T \times T \times T \times T \times ID \mid (Vt^- = T_{max} \vee Vt^+ < Vt^-) \wedge (Tt^- = T_{max} \vee Tt^+ < Tt^-) \}$$

where T_{max} represents Maximum values in the domain of T .

When replacing CT from Definition 1. with the approach where current facts or current tuples are represented as points with end times equal to start times, the *POINT* approach (Stantic et al. 2003), it follows:

Definition 3: The domain of Bitemporal tuples D^{POINT} on a domain of Timestamp values T and the domain of tuple identifiers ID for the *POINT* approach for representing current time can be defined as follows:

$$D^{POINT} \cong \{ \langle Vt^+, Vt^-, Tt^+, Tt^-, id \rangle \in T \times T \times T \times T \times ID \mid Tt^+ \leq Tt^- \wedge Vt^+ \leq Vt^- \}$$

In Bitemporal databases time evolves discretely. When an object is first inserted, its transaction time temporal attribute has the form $[Tt^+, now)$ indicating that tuple is current and ending time is unknown. Updates are allowed to be made to the objects of the most recent version only. No modification to the past is allowed, as the past cannot be changed. Deletion is logical; basically there are no physical deletions in Bitemporal databases. When an object is deleted, its transaction temporal attribute is changed from $[Tt^+, now)$ to $[Tt^+, Tt^-)$ where Tt^- is actually the current time when the transaction is executed.

Tables 1, 2 and 3 provide sample Bitemporal data for the different representations of *now*. Table 1 uses

<i>ID</i>	<i>Name</i>	<i>Position</i>	$[Vt^+$	$Vt^-)$	$[Tt^+$	$Tt^-)$
1	<i>Megan</i>	<i>DBA</i>	21.08.2002	10.05.2004	16.02.2002	UC
2	<i>Stephan</i>	<i>Teacher</i>	23.07.2000	30.01.2001	01.07.2001	26.10.2002
3	<i>Mark</i>	<i>Admin</i>	22.03.2001	now	01.07.2001	UC
4	<i>Steven</i>	<i>Officer</i>	21.02.2002	now	13.04.2000	03.02.2001
...
...

Table 1: Sample *now-relative* Bitemporal data

<i>ID</i>	<i>Name</i>	<i>Position</i>	$[Vt^+$	$Vt^-)$	$[Tt^+$	$Tt^-)$
1	<i>Megan</i>	<i>DBA</i>	21.08.2002	10.05.2004	16.02.2002	31.12.9999
2	<i>Stephan</i>	<i>Teacher</i>	23.07.2000	30.01.2001	01.07.2001	26.10.2002
3	<i>Mark</i>	<i>Admin</i>	22.03.2001	31.12.9999	01.07.2001	31.12.9999
4	<i>Steven</i>	<i>Officer</i>	21.02.2002	31.12.9999	13.04.2000	03.02.2001
...
...

Table 2: Sample Bitemporal data with *MAX* representation of *now*

<i>ID</i>	<i>Name</i>	<i>Position</i>	$[Vt^+$	$Vt^-)$	$[Tt^+$	$Tt^-)$
1	<i>Megan</i>	<i>DBA</i>	21.08.2002	10.05.2004	16.02.2002	16.02.2002
2	<i>Stephan</i>	<i>Teacher</i>	23.07.2000	30.01.2001	01.07.2001	26.10.2002
3	<i>Mark</i>	<i>Admin</i>	22.03.2001	22.03.2001	01.07.2001	01.07.2001
4	<i>Steven</i>	<i>Officer</i>	21.02.2002	21.02.2002	13.04.2000	03.02.2001
...
...

Table 3: Sample Bitemporal data with *POINT* representation of *now*

Variables to represent current time, whereas Tables 2 and 3 use the *MAX* and *POINT* approach respectively.

Because Bitemporal databases are basically append only, they are usually very large in size. Without effective management the informational benefits of Bitemporal data can be easily outweighed by the costs of poor access times (Dyreson, Snodgrass & Freiman 1995). For that reason efficient access methods are even more important in Bitemporal databases than in conventional databases.

3 Temporal Access Methods

A lot of multidimensional access structures have been proposed and some of them have been recommended for temporal databases (Kumar, Tsotras & Faloutsos 1997). The effectiveness of the majority of the proposed models have only been analysed on theoretical calculations based on worst case scenarios (Salzberg & Tsotras 1999). Despite *now-relative* data being a natural part of Bitemporal databases, only a few access methods have addressed this issue.

In this section we briefly review the relevant access methods for indexing *now-relative* Bitemporal data. Further we provide a review of some commercial Spatial access techniques, which could be used for Bitemporal databases, considering that valid time and transaction time are orthogonal and can be represented in two dimensional space.

3.1 Indexing *now-relative* data

In the literature it is generally agreed that usage of the Spatial indexes based on R-trees for Bitemporal data is possible and that the maximum-timestamp approach is a straightforward solution, i.e. the *MAX* approach (Bliujute et al. 1998). Further it is usually suggested as obvious that *now-relative* facts in the *MAX* approach are represented using very large rectangles, and that the resulting search performance

should be poor due to excessive dead space in the index nodes and overlap between nodes.

Problems with having variables, to represent current time, in index structures have been identified in the literature and several index structures that avoid variables have been proposed.

The GR-Tree (Bliujute et al. 1998) and the 4R-Tree (Bliujute et al. 2000) support both *now-relative* valid and transaction time. Essentially both seek to improve the efficiency of the underlying structures by reducing dead space in the tree formed by using the *MAX* approach in a standard R*-Tree. The GR-Tree extends the R*-Tree to offer storage for both static tuples (i.e. tuples with closed valid and transaction time ranges) and growing tuples (i.e. tuples with valid and/or transaction time end unknown, thus representing *now-relative* data). It does so by extending the domain of acceptable timestamp values to include the variables *now* and *UC* to represent current time in valid time and transaction time respectively. Further introducing two Flags, *Hidden* and *Rectangle*, in the non-leaf nodes where *Hidden* is used to represent growing geometries that may be placed together with other regions in a MBR (Maximum Bounding Regions) that it would eventually overgrow, while *Rectangle* is used to indicate bounding areas that are rectangles so as to be able to accommodate non-rectangle growing regions, thus helping reduce dead space and overlap.

To achieve a similar reduction in dead space, and to facilitate the use of an off-the-shelf indexing solution, the 4R-Tree transforms the growing tuples into points and lines. It identifies that growing rectangles cannot be formally implemented using an existing or general Spatial index such as an R-Tree or R*-Tree and thus offers to transform the growing rectangles in the following manner to make them indexable using a simple *off-the-shelf* indexing mechanism. Growing rectangles are mapped into a closed line while growing stairs are mapped into points. Also, to avoid overlap, the tuples are split into 4 R-Trees depending on whether valid time and transaction time are current

or not. It consequently also requires the queries to be transformed using the same logic before they can be implemented. It provides a representation of the data and query transformations for each of the R-Trees. It is claimed that the transformation of Bitemporal shapes into intervals, and especially points, instead of the normal enlarged rectangles, may positively impact performance as these simple shapes take up less space and would thus lead to trees with higher fanouts of the nodes. Also, though the index offers the possibility of using ‘off-the-shelf’ indexing solutions, it requires an implementation layer on top of the four R*-Trees to apply the insertion, deletion and query algorithms. This approach requires a query transformations and at the same time requires some modifications to the kernel, so off-the-shelf R-tree spatial indexes can not be simply reused.

In our approach, we logically divide the regions and additionally employ the geometric power of spatial indexes to perform query transformations without requiring modification to the kernel, so off-the-shelf Spatial R-tree indexes can be used. Further the *POINT* approach impacts positively on the performance.

3.2 Spatial Support offered by Commercial Databases

Some major commercial database vendors have introduced Spatial indexing while others are still working on its support. Though it is the most popular, not all database products offering Spatial support use R-Tree indexing for spatial applications. Another option offered by some commercial systems is to build a spatial index outside the server and store it in blobs. The index can then be accessed from middleware and blobs retrieved as needed. However, regardless of how good the index structure is, the need to retrieve and maintain blobs, containing portions of the index from outside the server, results in substantial I/O overhead and concurrency problems that make it slow and inefficient.

IBM’s DB2 uses a 3-stage Grid indexing method for its Spatial Extender. Unlike the R-tree, which is a data partitioning strategy, the Grid index (like the QuadTree) is a space partitioning method, where space is partitioned into cells, according to a regular grid or other hierarchical structure (e.g., a quad-tree). Each cell is assigned a number, and each spatial object is then associated with the number(s) of the cells it overlaps. But space is not a linear sequence, and the irregular nature of many spatial features requires complex queries and a large number of false hits that have to be examined. Essentially, compared to R-Trees, this proves to be a slow, inefficient method.

Sybase offers spatial support through Boeing Autometrics Spatial Query Server, though the kind of indexing used is claimed to be one of a kind and essentially targeted at Data Warehousing solutions.

The Oracle R-Tree is a combination of multiple techniques but it is called R-Tree for simplicity. The algorithms for Insert, Delete and Query are essentially based on the R*-Tree Algorithm while the Creation algorithm is a combination of many techniques including R-Tree, TGS, VAMSplit, STR etc. The query Algorithm also takes advantage of many unpublished optimization techniques. Each MBR (Maximum Bounding Region) is stored only once in the index entries and thus provides effective storage (and is therefore an R-Tree / R*-Tree structure and not an R+ -Tree Structure).

The natural choice for a platform for empirical testing carried as part of this research is Oracle 9i. This is because Oracle offers R*-Tree indexing, the

most popular proposed Spatial index for Bitemporal data so far.

4 The *POINT* Approach

In this section we define the queries chosen to measure the query performance of the Spatial indexes on *now-relative* Bitemporal data and give a query transformations for the *POINT* approach. Further we extend the query notation to better define range and timeslice queries.

4.1 Query Notation

The notation presented in (Tsotras, Jensen & Snodgrass 1998) represents a unifying approach for referring to Spatiotemporal queries.

The basic notation used to classify Spatiotemporal queries is as follows:

Key // Spatial Attributes // Temporal Attributes

Where “//” is used to distinguish the Explicit, Spatial and Temporal attributes as inspired by (Kleinrock 1975). The extended form of the same could be represented as:

Key // X_ dimension/Y_ dimension/Z_ dimension // Valid/Transaction

From this, the notation of interest to this study, i.e. the Bitemporal Query, can be derived as follows:

$\langle \text{Bitemporal Query} \rangle = \text{Key} // \langle \text{VTQ} \rangle / \langle \text{TTQ} \rangle$

where, *VTQ* and *TTQ* represent the Valid Time Qualifier and Transaction Time qualifier respectively. The various values that can be assigned to the Temporal Qualifiers are as follows:

- “*V*” represents a single attribute value and corresponds to a single time instant.
- “*R*” represents a specified range of attribute values and corresponds to a continuous time interval.
- “*S*” represents a set of ranges of attribute values and corresponds to a set of specific time intervals or a set of specific time instants.
- “***” indicates that any value is acceptable in this entry to satisfy the query.
- “*-*” indicates that the data does not include this qualifier, making this entry inapplicable.

A part of classifications of interest for this study is provided in Table 4(Tsotras et al. 1998). This classification was proposed to provide a unifying approach to referring to spatiotemporal queries.

Proposed Name	Notation
Bitemporal pure-timeslice	<i>*/V/V</i>
Bitemporal range-timeslice	<i>R//V/V</i>
general Bitemporal query	<i>R//R/R</i>

Table 4: Query Notation for some Queries

We extend the query notation so it can better indicate the actual range of interest in particular Range queries by including the starting and ending points into the notation. For example Range query stating from time zero, which represent the minimum value in time Domain *T*, up to *now* is represented as:

$$R_0^{now}$$

Similarly query that timeslices the time line at the time instant *history* is represented as:

$$V_{history}$$

4.2 Query Transformations

A number of queries are chosen to measure the query performance of the Spatial indexes on *now-relative* Bitemporal data. The queries are Spatial representations of the Bitemporal timeslice queries proposed in (Torp, Jensen & Snodgrass 1997) and used in (Stantic et al. 2003) to measure the performance of the *MAX* and *POINT* representations in the non-Spatial domain. Fig 1 represents the temporal semantics of the queries.

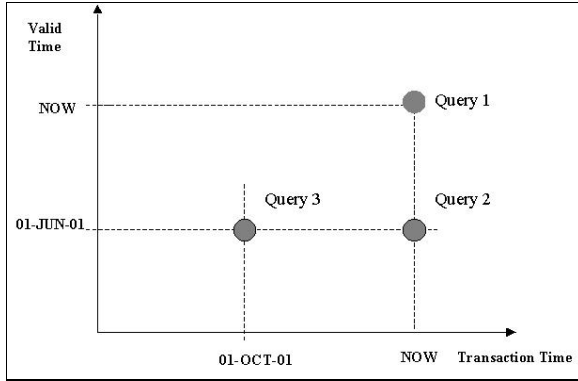


Figure 1: Time Slice Queries Q1, Q2, and Q3

Query 1 retrieves the current state in both transaction time and valid time. Semantically, it retrieves our current belief about tuples valid in the current state of the modelled reality. It selects tuples with valid time intervals that overlap with *now* and transaction times that meet *now*. It therefore selects tuples that have valid time intervals starting at or before *now* and ending after *now* and transaction time end equal to *now*.

For Query 1 the *MAX* approach timeslice query notation would be $*/V_{now}/V_{T_{max}}$ and it represents domain of tuples that meets following criteria:

$$[Vt^+ \leq now \wedge Vt^- > now \wedge Tt^- = T_{max}]$$

Spatially transformed data from table 2 are shown in Figure 2, values for $Vt_1 \dots Vt_6$ represent the valid time values from table 2, similarly for transaction time values $Tt_1 \dots Tt_5$. A geometry that would have $[Vt^+ \leq now \wedge Vt^- \geq now \wedge Tt^- = T_{max}]$ would have to intersect the point $[now, T_{max}]$.

For Query 1 using the *POINT* approach represents the domain of tuples that meets the following criteria:

$$[(Vt^+ \leq now \wedge (Vt^- > now \vee Vt^- = Vt^+)) \wedge Tt^- = Tt^+]$$

The Spatial transformation of the *POINT* representation for Query 1 as shown in Figure 3 is a little more complex. Since we look for all tuples where transaction time is current $Tt^- = Tt^+$, which means that tuples of interest for this query can be represented as points or lines only, rectangles would not be part of the result set.

Shape	Geometry Type
Point	Type 1
Line	Type 2
Rectangle	Type 3

Table 5: Spatial Geometry Types of interest

Tuples where $[Vt^+ \leq now \wedge Vt^- = Vt^+]$ would be points (since $Tt^- = Tt^+$) and would lie below the line

$[VT = now]$. Also, geometries where $[Vt^+ \leq now \wedge Vt^- > now]$ would be line geometries, parallel to the VT-Axis since $Tt^- = Tt^+$, which intersect the line $[VT=now]$. The *POINT* query thus returns all tuples representing point geometries that intersect with the area under the line $[VT=now]$ and line geometries parallel to the VT-Axis intersecting with $[VT=now]$. For the *POINT* approach the current time slice query notation would be:

$$(*//R_0^{now}/R_0^{now}) \cup (*//V_{now}/R_0^{now})$$

and it represents a union set of the points and lines from the previously mentioned regions.

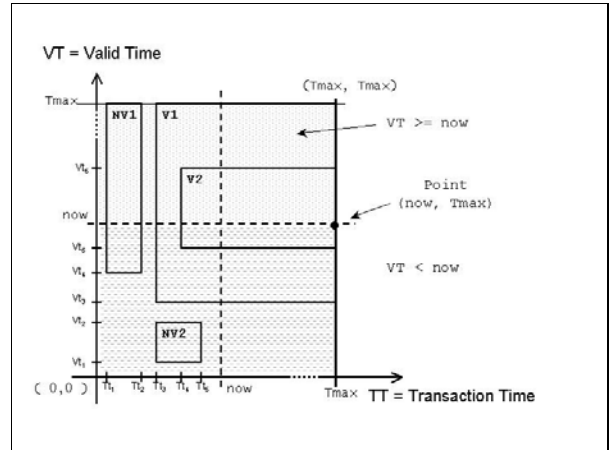


Figure 2: Query 1: Spatial *MAX* Approach

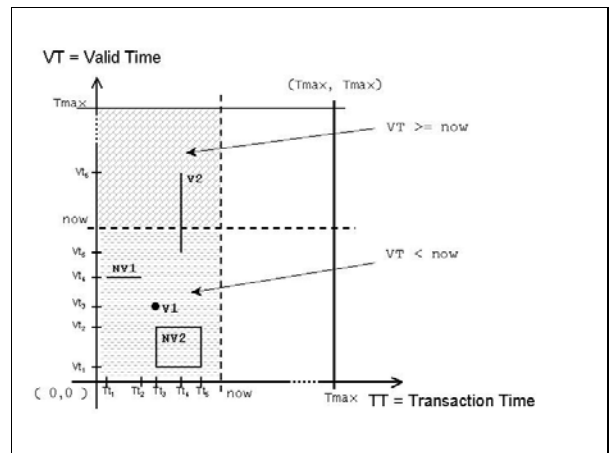


Figure 3: Query 1: Spatial *POINT* Approach

Figures 2 and 3 represent the semantics of Query 1 with tuples representing geometries $[V1, V2]$ being returned as part of the result set while tuples representing geometries $[NV1, NV2]$ are rejected.

Query 2 timeslices the relation as of current time in the transaction time domain and as of a past time in the valid time domain. Semantically, it retrieves our current belief about a past state of the world. In the non-Spatial approach, the query condition for the *MAX* approach is represented by:

$$[Vt^+ \leq history \wedge Vt^- > history \wedge Tt^- = T_{max}]$$

while for the *POINT* approach query condition is:

$$[Vt^+ \leq history \wedge (Vt^- > history \vee Vt^- = Vt^+) \wedge Tt^- = Tt^+]$$

The Spatial representation for Query 2 is somewhat similar to Query 1 and geometries satisfying the query in the *MAX* approach would have to intersect the point (history, T_{max}). The Representation for the query would thus be: $[*//V_{history}/V_{T_{max}}]$

Similarly, for the *POINT* approach query would be satisfied by all *Point* geometries that intersect with the area under the line $[VT=history]$ and all *Line* geometries parallel to the VT-Axis intersecting with, but not ending at, $[VT=history]$. For the *POINT* approach history time slice query notation would be:

$$(*//R_0^{history}/R_0^{now}) \cup (*//V_{history}/R_0^{now})$$

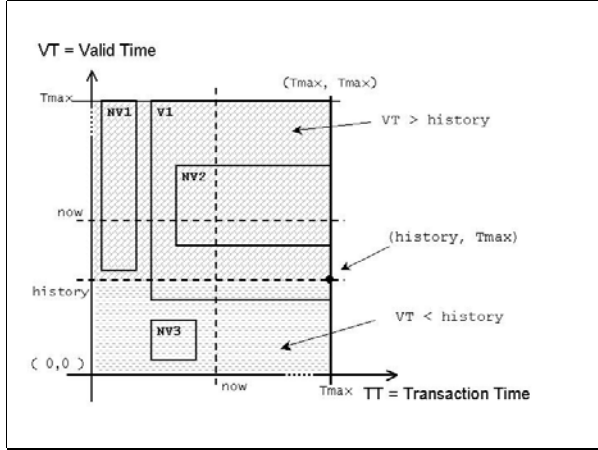


Figure 4: Query 2: Spatial *MAX* Approach

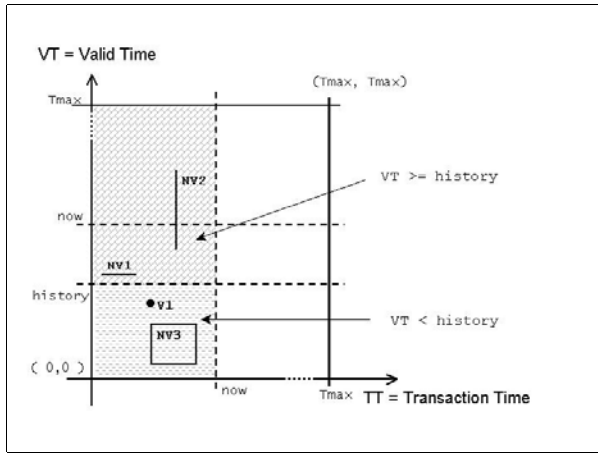


Figure 5: Query 2: Spatial *POINT* Approach

Figures 4 and 5 represent the semantics of Query 2 with the tuple representing geometry $[V1]$ being returned as part of the result set while tuples representing geometries $[NV1..NV3]$ are rejected.

Query 3 timeslices the relation as of a past time(history) in both the transaction time and valid time domains. Semantically, it retrieves our past belief about a past state of the world. In the non-Spatial approach, the *MAX* query condition would therefore be represented as:

$$[Vt^+ \leq history \wedge Vt^- > history \wedge Tt^+ \leq history \wedge Tt^- > history]$$

while for the *POINT* representation:

$$[Vt^+ \leq history \wedge (Vt^- > history \vee Vt^- = Vt^+) \wedge Tt^+ \leq history \wedge (Tt^- > history \vee Tt^- = Tt^+)]$$

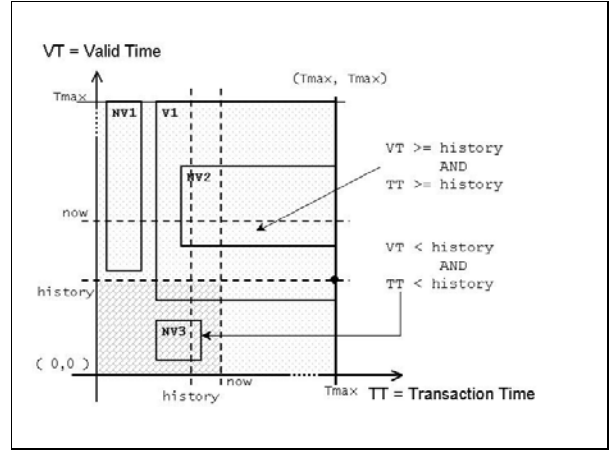


Figure 6: Query 3: Spatial *MAX* Approach

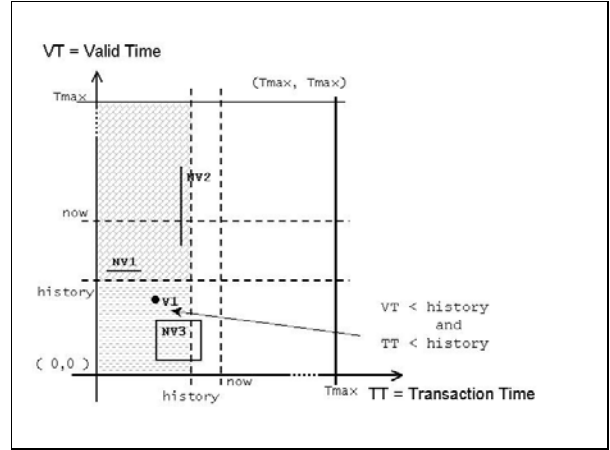


Figure 7: Query 3: Spatial *POINT* Approach

Spatially geometries satisfying Query 3 in the *MAX* representation would have to pass through the point (history, history) and would be represented as a $[*//V_{history}/V_{history}]$ query. Similarly, the *POINT* query would be satisfied by all *Point* geometries lying in the rectangle (0, history, 0, history), all *Line* geometries parallel to the VT-Axis intersecting with, but not ending at, $[VT=history]$, all *Line* geometries parallel to the TT-Axis intersecting with, but not ending at, $[TT=history]$ and all *Rectangle* geometries intersecting with the point (history, history).

Figures 6 and 7 represent the semantics of Query 3 with the tuple representing geometry $[V1]$ being returned as part of the result set while tuples representing geometries $[NV1..NV3]$ are rejected.

The large rectangles of current tuples in the *MAX* approach cause index entries to be spread across many nodes. In contrast the *POINT* approach can store many current tuples index entries in the same node. In addition the *POINT* approach enables current entries (long lived) to be stored efficiently with short lived entries.

The *POINT* approach has a smaller total area of MBR needed to cover the same tuples and needs relatively fewer leaf nodes, which reduces the total number of disk accesses required to answer the query resulting in better performance, as is shown in Pictures 8 and 9.

In order to evaluate the efficiency and accuracy of our logical query transformations combined with spatial index geometry features, we decided to empirically compare the *POINT* to the *MAX* approach.

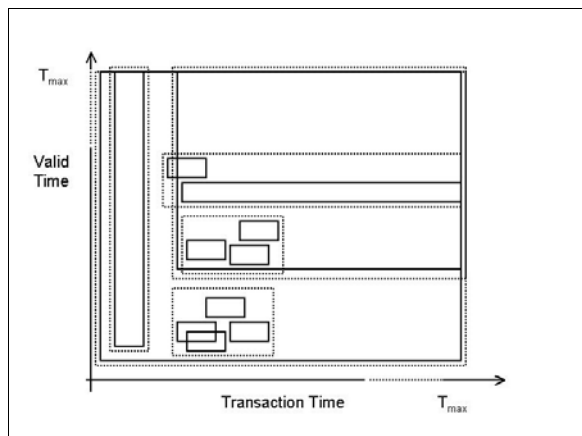


Figure 8: MBR for MAX approach

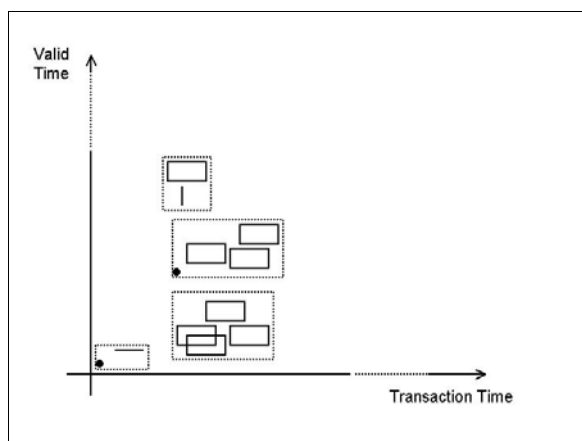


Figure 9: MBR for POINT approach

5 Experiment

No empirical results could be found to confirm the poor performance of the *MAX* approach to indexing *now-relative* Bitemporal data as is highlighted in the literature. So we performed tests of Spatial indexes on real Bitemporal data in a real environment and compared the results with the results of the *POINT* approach tested in same environment.

The Implementation shall:

- Create Spatial indexes on *now-relative* Bitemporal data using both *MAX* and *POINT* approaches to represent *now*. Each approach will have three relations with one million randomly generated tuples, having 10%, 20% and 40% of the tuples overlapped with the current time in both Transaction and Valid time domains.
- Use the most popular, and most widely accepted Spatial access method proposed in literature, the R*-Tree index.
- Do so completely within the scope of an existing commercial RDBMS, Oracle 9i - version 9.2.0.1.0.
- Not require any modification to the database kernel nor any additional implementation layer.

5.1 Measurables

There needs to be certain criterion against which the implementation can be assessed. To the best of our knowledge this is the first such an implementation, so quantitative evaluation is possible only within the

various result sets obtained from the implementation. For the qualitative assessment and quantitative evaluation, we shall use the following criteria proposed by Gaede and Gunther (Gaede & Gunther 1998):

Dynamics: The ability to keep track of changes as objects are inserted and deleted from the database in any order.

Broad Range of Supported Operations : Ability to efficiently support Insert, Update, Delete, and Query processes.

Simplicity: Ability to integrate easily and work efficiently in large-scale applications.

Time Efficiency: Ability to perform fast searches as well as inserts, updates and deletions.

Space Efficiency: Especially as compared to the data that it indexes.

Concurrency and Recovery: Ability to efficiently manage concurrent access and support recovery operations.

Minimum Impact: Ability to integrate well into an existing system with minimum impact on existing parts of the system.

Our recommendation doesn't require any modification to the database kernel or additional implementation logic and it is within the existing capability of commercial RDBMS, which ensures the Broad Range of Supported Operations (Insert, Update, Delete), Concurrency and Recovery, and to a large extent Dynamics, so these criteria can be ruled out and are not be considered.

Since we consider different distributions of *now-relative* data, the dependance of performance based on different data sets, especially different amounts of *now-relative* data is also studied.

The Theory of Indexability (Hellerstein, Koutsoupias & Papadimitriou 1997) identifies I/O complexity cost, measured by the *number of disk accesses*, as one of the most important factors for measuring query performance. The *Oracle 9i Performance Tuning Guide* (Green 2002) also establishes the importance of Disk Scans (physical I/O) and terms reduction of this I/O overhead as an extremely important performance goal. Disk Reads, also called Disk Accesses or Disk Scans, are thus used to assess the Query performance. Other measures of importance such as *CPU usage* are also used in conjunction with the number of Disk Accesses to assess the performance of the Create, Insert, Update and Delete processes.

To ensure no effect of environment parameters on the results, the Oracle Instance is tuned appropriately and parameters such as SGA (System Global Area) size and tolerance are varied to study their effect on accuracy, R-Tree structure and performance and results will be mentioned briefly due to space limitation.

5.2 Environment

The implementation is carried out on a four 450MHZ CPU - SUN UltraSparc II processor machine, running Oracle 9.2.0.1.0 RDBMS, with a database block size of 8K using Oracle Spatial 9i Release 2 (Murray 2002), hereon referred to as Spatial.

The Spatial component of a Spatial feature is the geometric representation of its shape, referred to as its geometry and stored in the Spatial data type - MDSYS.SDO_GEOMETRY, which acts as a container for storing points, lines and polygons. Attributes consist of a geometry

type identifier (SDO_GTYPE), a Spatial reference system identifier (sdo_srid), an element descriptor array (sdo_elem_info), and an ordinate array (sdo_ordinates) among others. Sdo_ordinates contains the values for coordinate pairs or triples that define the vertices of the geometric elements. Sdo_elem_info defines how these ordinates should be assigned to the element or elements that constitute the geometry.

Spatial offers two types of indexing techniques, Quad-Tree indexing and R-Tree indexing. *The Oracle Spatial R-Tree is essentially an Optimised R*-Tree structure.* The algorithms for Create, Insert, Delete and Query are based almost entirely on the R*-Tree algorithm while the Create and Query algorithm include packing optimisation using the TGS, VAMSplit, STR and some unpublished algorithms among others. Oracle 9i introduced support for Function-based Spatial indexes, used in this implementation. Oracle 9i Release 2, added functionality to allow Parallel creation of R-Trees and boasts Performance enhancements of up to 50% in Index creation and up to 40% with Spatial queries using Secondary filters.

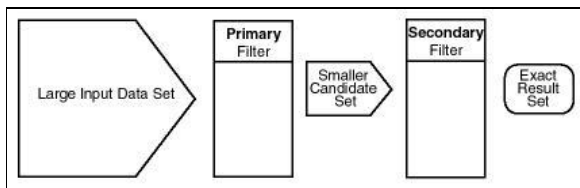


Figure 10: Oracle Spatial: Query Model

Spatial uses a two-tier Query Model to resolve Spatial queries, as shown in Fig 10 (Murray 2002). The first operation, called Primary Filter which is implemented using the SDO_FILTER function. It is a low cost filter that compares geometry approximations to return a superset of the exact result set. The second operation, called Secondary Filter, which is implemented using the sdo_relate function is computationally expensive and is applied to the result set of the primary filter to yield an accurate answer to a Spatial query. The R-tree index is responsible mainly for the primary filter while the Secondary Filter combines R-Tree based pruning (SDO_FILTER) and geometry-geometry comparisons. Thus, the performance of both filters is studied as part of the implementation.

A method of the SDO_GEOMETRY type provided by Spatial and used in the implementation to exploit the computational advantage of the POINT approach, is the get_gtype method which takes the geometry as input and returns the geometry type as per the sdo_gtype parameter.

Sample code for Query 1 for both approaches is provided below:

```
@read_start_parameters
select
  count(position) from max10 s
where
  (sdo_filter(get_max_geo(
    s.Vts, s.Vte, s.Tts, s.Tte),
    get_max_ge(sysdate, sysdate,
      '31-DEC-9999', '31-DEC-9999'),
    'querytype = WINDOW' )='TRUE'
  );
@read_end_parameters
```

This query finds all geometries that intersect with the point (sysdate, '31-DEC-9999'). It uses the function get_max_geo which takes coordinates (Vt^+ , Vt^- , Tt^+ , Tt^-) as input returns the corresponding geometry of sdo_geom type.

```
@read_start_parameters
select
  count(position) from point10 s
where
  ((get_same_geo(s.Vts, s.Vte,
    s.Tts, s.Tte).get_gtype()=1
  AND sdo_filter(get_same_geo(
    s.Vts, s.Vte, s.Tts, s.Tte),
    get_same_geo(
      0, sysdate, 0, sysdate),
    'querytype = WINDOW' )='TRUE')
  OR
  (get_same_geo(s.Vts, s.Vte,
    s.Tts, s.Tte).get_gtype()=2
  AND sdo_filter(get_same_geo(
    s.Vts, s.Vte, s.Tts, s.Tte),
    get_same_geo(
      sysdate, sysdate, 0, sysdate),
    'querytype = WINDOW' )='TRUE')
  );
@read_end_parameters
```

This query finds all point geometries (type 1) that are within region (0, sysdate,0,sysdate) and all lines geometries (type 2) that intersect with the line defined by (sysdate, sysdate,0, sysdate) it uses the function get_same_geo, which takes coordinates, attribute of the table point10 (Vts, Vte, Tts, Tte), as input and returns the corresponding geometry type while taking care that different geometry types i.e. points, lines and rectangles, are explicitly defined. As mentioned earlier rectangle geometry (type 3) are not of interest for Query 1. Full query statements and scripts used for population of data and index creations can be found (Khanna 2003).

A series of problems were encountered during the implementation and analysis, such as undocumented memory bugs in Spatial and unexplained structural details critical for analysis. A lot of support was sought and provided from Oracle staff. Some of these solutions were documented for the benefit of future research, and can be found at (Khanna 2003).

5.3 Analysis

The Query Performance results for (Query 1), see Fig 11 and 12, proved the earlier hypothesis that the straightforward maximum-timestamp, i.e. MAX approach, is inefficient in the handling of *now-relative* data by providing the first empirical proof. The inefficiency increases with growing amount of *now-relative* data. The POINT approach however, proved to be extremely efficient for the handling of *now-relative* data, with little or no effect from growing *now-relative* data, **to the extent of outperforming the Spatial MAX approach by over a factor of 20 in number of Disk Accesses, which is considered to be the most important.**

Better performance of the POINT approach is achieved by reducing the dimensions of the Spatial geometries, typically rectangles representing *now-relative* data, are represented as lines or points, which take up less room in the leaf nodes than the original rectangles, leading to higher fanout. The smaller dimensions of these geometries means there is less overlap between the maximum bounding regions, which form the boundaries of each node. In addition, because the representation of *now-relative* data in general does not extend beyond the current time, the amount of dead space (nodes that are searched that do not contribute to the answer) is reduced. Another explanation for the better performance of the POINT approach is its reduction of the search space, due to logically dividing the total space to the areas of inter-

est and identifying only the geometry types of interest, which improves the performance significantly.

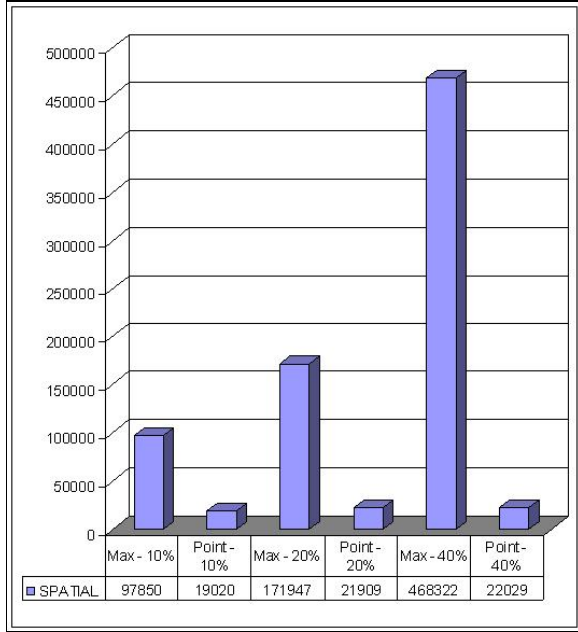


Figure 11: Query 1: Disk Accesses - Secondary Filter

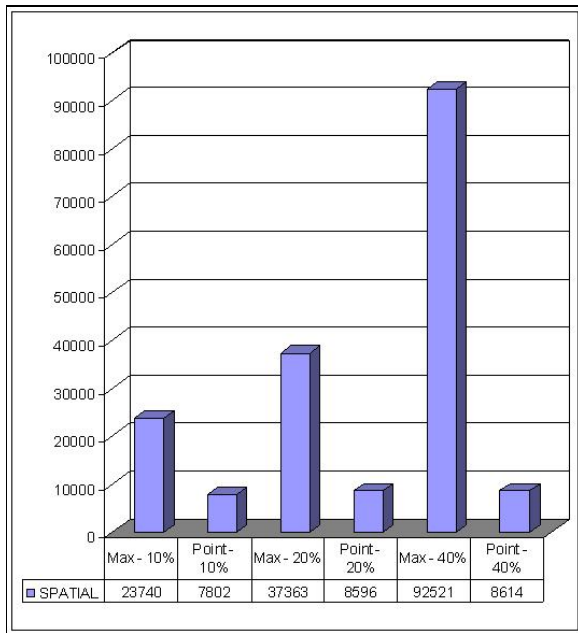


Figure 12: Query 1: CPU Usage - Secondary Filter

Approach	percentage	Disk Accesses	CPU
Max	10%	138147	34111
Point	10%	127030	45071
Max	20%	157983	27681
Point	20%	67598	42462
Max	40%	186681	22925
Point	40%	30507	38605

Table 6: Query 2: Disk Accesses and CPU usage

Experiment results for Queries 2 and 3, shown in Tables 6 and 7, demonstrate that the *POINT* approach also has a better performance in term of Disk accesses. Its relative performance, compared to the *MAX* approach, improves with larger amount of *now-relative* data. This is due to the same reasons as

Approach	percentage	Disk Accesses	CPU
Max	10%	180353	30291
Point	10%	142478	47740
Max	20%	165980	29538
Point	20%	92560	65366
Max	40%	75038	12134
Point	40%	35155	31374

Table 7: Query 3: Disk Accesses and CPU usage

for Query 1, by reducing both the maximum bounding regions, dead space, and a logical reduction of search space. However the CPU usage is worse for the *POINT* approach due to complex logical query transformations.

In addition to the Select command, we tested performance of Create, Update and Delete commands for both approaches using different amounts of *now-relative* data. The Update performance, though not affected by the amount of *now-relative* data, was greatly affected by the representation used. Similarly for the Create commands performance, the *MAX* approach consistently proved to be twice as computationally expensive in terms of CPU Reads and significantly more expensive in terms of Disk Accesses.

We could not identify any significant influence of parameters, such as size and tolerance of SGA, on the performance of the two method tested.

The space used for the *POINT* approach index structure is significantly smaller, as is the number of Disk Accesses and CPU usage during the initial index creation.

6 Conclusion and Future Work

This study makes the following contributions to the field:

- We presented a method to efficiently index *now-relative* Bitemporal data by using the *POINT* approach, and
- empirically demonstrated it to have an improvement, outperforming the Spatial MAX approach by over a factor of 20.
- we proved the previous hypothesis that straightforward maximum-timestamp approach has poor performance;
- the proposed method is based only on logical transformations of queries without the need for modification to the kernel, so an off-the-shelf Spatial index can be used;
- we extended the Spatiotemporal Query notation for Bitemporal databases, to better define range and timeslice queries;
- we defined the domain of Bitemporal Tuples for different representations of *now*;
- we surveyed existing proposed Spatial Access methods to assess their effectiveness for indexing Bitemporal Data including *now-relative*.
- we identified the type and level of support offered by Commercial Databases for indexing Spatial data.

The identified advantages of the *POINT* approach, to index *now-relative* Bitemporal data, can be employed to create a new more efficient access method.

It would be interesting to investigate the effectiveness of the *POINT* approach on other types of queries of interest, such as interval queries.

References

- Bliujute, R., Jensen, C. S., Saltenis, S. & Slivinskas, G. (1998), R-Tree Based Indexing of Now-Relative Bitemporal Data, in 'VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, New York, USA', pp. 345–356.
- Bliujute, R., Jensen, C. S., Saltenis, S. & Slivinskas, G. (2000), Light-Weight Indexing of General Bitemporal Data, in 'Statistical and Scientific Database Management', pp. 125–138.
- Clifford, J., Dyreson, C., Isakowitz, T., Jensen, C. S. & Snodgrass, R. T. (1997), 'On the semantics of "Now" in databases', *ACM Transactions on Database Systems (TODS)* **22**(2), 171–214.
- Date, C., Darwen, H. & Lorentzos, N. (2002), *Temporal Data and the Relational Model*, Morgan Kaufmann.
- Dyreson, C. E., Snodgrass, R. T. & Freiman, M. (1995), Efficiently Supporting Temporal Granularities in a DBMS, Technical Report TR 95/07. *citeseer.nj.nec.com/dyreson95efficiently.html
- Gaede, V. & Gunther, O. (1998), 'Multidimensional Access Methods', *ACM Computing Surveys (CSUR)* **30**(2), 170–231.
- Green, C. D. (2002), 'Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2) Part No. A96533-02'. *http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96533/title.htm
- Hellerstein, J., Koutsupias, E. & Papadimitriou, C. (1997), 'On the Analysis of Indexing Schemes', *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.
- Jensen, C. S. (2000), 'Introduction to temporal databases, research', <http://www.cs.auc.dk/csj/Thesis/pdf/chapter1.pdf>.
- Jensen, C. S. & Snodgrass, R. (1999), 'Temporal Data Management', *IEEE Transactions on Knowledge and Data Engineering* **11**(1), 36–44.
- Khanna, S. (2003), 'Personal communication with Oracle'. *<http://miami.int.gu.edu.au/POINT/comm.htm>
- Kleinrock, L. (1975), *Queueing Systems: Theory*, Vol. 1, 1 edn, John Wiley and Sons.
- Kumar, A., Tsotras, V. J. & Faloutsos, C. (1997), 'Designing access methods for bitemporal databases', *University of Maryland at College Park; Report No. UMIACS-TR-97-24*.
- Murray, C. (2002), 'Oracle Spatial User's Guide and Reference, Release 9.2 Part No. A96630-01'. *http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96630/title.htm
- Salzberg, B. & Tsotras, V. J. (1999), 'Comparison of Access Methods for Time Evolving Data', *ACM Computing Surveys* **31**(1).
- Snodgrass, R. & et al. (1987), 'The Temporal Query Language TQEL', *ACM TODS* **12**(2), 247–298.
- Snodgrass, R. & Ahn, I. (1986), 'Temporal databases', *IEEE Computer* **19**(9), 35–42.
- Snodgrass, R. T. (2000), *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann.
- Stantic, B., Thornton, J. & Sattar, A. (2003), 'A Novel Approach to Model NOW in Temporal Databases', In *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning (TIME-ICTL 2003)*, Cairns pp. 174–181.
- Torp, K., Jensen, C. S. & Snodgrass, R. T. (1997), 'Stratum Approaches to Temporal DBMS Implementation', *A TIMECENTER Technical Report*.
- Tsotras, V. J., Jensen, C. S. & Snodgrass, R. T. (1998), 'An Extensible Notation for Spatiotemporal Index Queries', *ACM SIGMOD Record* **27**(1), 47–53.