

A UNIFYING FRAMEWORK FOR TEMPORAL DATABASES

Daniel Thomas
Algorithms For Database Systems, SS2010

CONTENTS

1. Introduction	3
2. Temporal Data, Databases and Query Languages	3
3. A Unifying Formal Framework	4
3.1 Defining The Temporal Domain	5
3.2 The Abstract Temporal Database	6
3.2.1 The Module Theoretic View	6
3.2.2 The Snapshot View	7
3.2.3 The Timestamp View	8
3.3 Abstract Temporal Query Languages	8
4. Applying The Framework To TSQL2 And HRDM	9
4.1 TSQL2	9
4.2.1 A Short Introduction To TSQL2	9
4.2.2 The Temporal Domain Of TSQL2	9
4.2.3 The Abstract Temporal Database Implemented By TSQL2	10
4.2.4 The Abstract Temporal Query Language Of TSQL2	10
4.2 HRDM	10
4.2.1 A Short Introduction To HRDM	10
4.2.2 The Temporal Domain Of HRDM	11
4.2.3 The Abstract Temporal Database Implemented by HRDM	11
4.2.4 The Abstract Temporal Query Language Of HRDM	11
4.3 Comparison Of the Abstract Properties Of TSQL2 And HRDM	11
5. Conclusions And Criticism	13
Bibliography	14

A UNIFYING FRAMEWORK FOR TEMPORAL DATABASES

Daniel Thomas
Algorithms For Database Systems, SS2010

1. Introduction

Much research has been done on temporal databases and query languages. Also, many papers have been written on this subject, as listed (slightly outdated) in [1]. There is even a „Symposium on Spatial and Temporal Databases (SSTD)“[2] which was held last in 2009. However, most papers referenced in the last few iterations of this symposium are concerned with spatial data and treat the temporal aspect only when it is relevant for spatiotemporal databases. Recent papers concerned *purely* with temporal databases are hard to find.

This report focuses on a unifying framework for temporal databases, as proposed in [3]. Chapter 2 gives a short introduction on temporal data, temporal databases and their query languages and tries to convey the need for temporal databases. In Chapter 3, the formal framework for classification of temporal databases and query languages is introduced which is used in Chapter 4 to characterize the TSQL2 Query Language as well as the Historical Relational Data Model (HRDM) and subsequently compares them. Chapter 5 contains conclusions as well as criticism of the unifying framework.

2. Temporal Data, Databases And Query Languages

Temporal information occurs in many applications, such as:

- Government agencies storing information about citizens e.g. date of birth, when the driving license was issued or in which town a citizen lived at a given time.
- Financial transactions may be stored together with the date and time they were made.
- Telecommunication Companies may store the time and duration of every issued phone call, for billing purposes.

As a short, motivating example one could look at a database storing a small profile of every person in a country. One could model such a relation to contain following attributes: the social security number as key, the name, current hometown as well as the current occupation for every citizen. While the social security number may stay the same over time, all other attributes are likely to change. This raises many questions: what happens if, for example, the current hometown of a citizen changes? A simple solution would be to just update the corresponding tuple, but this inevitably leads to a loss of data, as it effectively deletes the information on the former hometown. Also, what happens when a citizen dies? Is the tuple just deleted from the database? What about queries like „list all people that lived in Zürich in 1999“ ?

Standard, non-temporal databases have serious shortcomings when handling such information. When facts change, data may be overwritten leading to the loss of potentially important, historical data. Temporal databases and their associated query languages try to make it easy to handle such data by accompanying data with temporal information and providing means for working with collections of such data in an intuitive way.

There are many different ways of tackling the problem of temporal databases and, consequently, many different databases and query languages have been proposed. Some of them are quite different in the way that they store and retrieve temporal data (e.g. TSQL2 and Backlogs), while others share many of their fundamental ideas (TSQL2 and TQuel). The following chapters try provide means for characterizing the abstract ideas of such database systems and languages so that one can easily categorize and compare them.

3. A Unifying Formal Framework



Figure 1: The Unifying Formal Framework (TDB: Temporal Database, TQL: Temporal Query Language)

In [3], Jan Chomicki introduces a formal framework in which he „attempted to extract the most important features of those [temporal] models and languages, and present them in a single unified framework.“ The well known ANSI-SPARC architecture, as described in [4],

defines three different level in a database system: the external level (the information from a users point of view), the conceptual level (what data is stored and how it is organized/ interrelated) and the physical level (how the data is stored physically). For temporal databases and query languages, we further split the conceptual level into abstract temporal databases and concrete temporal databases. Having done this, we can now take concrete temporal databases and query languages, and try to distill different characterizing properties out of them to be able to categorize and compare.

Figure 1 illustrates the basic idea: every concrete temporal database (which is a working implementation/description of a temporal database) *implements* at least one abstract temporal database, which corresponds to an abstract characterization of the concrete temporal database. The same is done for concrete and abstract temporal query languages. This chapter tries to show the different properties of a concrete temporal database and query language that could be considered important enough to be represented in their abstract representation. Three important properties are explained:

- The Temporal Domain
- The Abstract Temporal Database
- The Abstract Temporal Query Language

3.1 Defining The Temporal Domain

When talking about temporal databases and query languages, it is of greatest importance to define the *temporal domain*. There are different ways of defining the term temporal domain, a very simple temporal domain for example may have following definition:

„A time domain is a pair $(T, \leq)$ where T is a non-empty set of time instants and $\leq$ is a total order on T .“[5]

More complicated domains are conceivable, following distinct properties will help in characterizing them:

- instants vs. intervals: either one could work with time by defining an interval as a set of *time instants* (of some sort) or directly work with intervals. Most temporal databases and query languages have an instant based time model.

- dense vs. discreet: this is an important distinction, as semantically identical queries may produce different results depending on a dense or discreet temporal domain. For example, the a query for all persons with age under 24 will return all persons that are 23, 22 etc. if the database has a discreet temporal domain with granularity of years. On a temporal database with a dense temporal domain, the query will also return persons with an age between 23 and 24 years.
- time granularity: it is possible to define granules of time (identified by a set of time instants) so that the granules themselves are comparable. A good example would be to define a calendar (with weeks, months, years) over the set of days.
- temporal dimensions: it is possible to have multiple time axes on which facts may be defined. For example, some temporal databases define 2 time dimensions: valid time, defined as „the time when the fact is true in the modeled reality“[6] and transaction time which one can define as „the time when the fact is current in the database and may be retrieved.“[6]

3.2 The Abstract Temporal Database

In the abstract temporal database, one tries to capture the „representation independent meaning of a temporal database“[3]. Basically, Jan Chomicki identifies 3 different views on temporal databases, which eventually all should have the same expressive power. So the Tables 1, 2 and 3 all store the same information under different views.

3.2.1 The Module Theoretic View

This is how somebody with experience with non-temporal databases may model time in a relational database. Table 1 is an example of a relation in the model theoretic view. Basically, the module theoretic view just adds a temporal field to every tuple in a relation, specifying that the fact in this tuple is only true at the provided time instant. It is easy to see that, depending on the defined time instants, this can lead to an extremely big database. For example, in Table 1 time instants are years, so for every year a tuple is needed for every person in the database, regardless if anything has changed since the year before.

N R .	N A M E	H O M E - T O W N	O C C U P A T I O N	Y E A R
000-00-0000	Daniel	Wollerau	Student	1995
000-00-0000	Daniel	Wollerau	Student	1996
...	...	...	...	...
000-00-0000	Daniel	Pfäffikon	Student	2008
000-00-0000	Daniel	Dietikon	Student	2009
000-00-0000	Daniel	Dietikon	Student	2010
000-00-0001	Hans	Wollerau	Student	1995
000-00-0001	Hans	Zürich	Student	1996
...	...	...	...	...

Table 1: Model-Theoretic view on a simple temporal database containing information on the occupation and hometown of citizens.

3.2.1 The Snapshot View

In this view, for every time instant, all facts are stored that are hold for that instant. Table 2 illustrates this. Again, this leads to rather bloated tables, and also raises the question what happens if there are many years in which the tuples that are current for those years do not change. Is it really necessary to store them for every year separately?

Y E A R	P E R S O N
1995	{(000-00-0000, Daniel, Wollerau, Student), (000-00-0001, Hans, Zürich, Student)}
1996	{(000-00-0000, Daniel, Wollerau, Student), (000-00-0001, Hans, Zürich, Student)}
...	...
2008	{(000-00-0000, Daniel, Pfäffikon, Student), (000-00-0001, Hans, Zürich, Student)}
...	...

Table 2: Snapshot view on a simple temporal database containing information on the occupation and hometown of citizens.

3.2.1 The Timestamp View

Here, for every (data-)tuple a timestamp is stored which defines the set of time instants for which this fact holds true. Table 3 shows how such a timestamp view could look like. This is the preferred view of most temporal databases. The timestamps are formulas with at least one variable in the language of the temporal domain, making it easy to associate a fact (like that Hans lived in Zürich and was a student) with a timespan (with the timestamp formula only being true if t is between 1995 and 2011).

N R .	N A M E	H O M E - T O W N	O C C U P A - T I O N	Y E A R S
000-00- 0000	Daniel	Wollerau	Student	$1995 < t < 2008$
000-00- 0000	Daniel	Pfäffikon	Student	$2008 \leq t < 2009$
000-00- 0000	Daniel	Dietikon	Student	$2009 \leq t < 2011$
000-00- 0001	Hans	Zürich	Student	$1995 \leq t < 2011$

Table 3: Timestamp view on a simple temporal database containing information on the occupation and hometown of citizens.

3.3 Abstract Temporal Query Languages

Now it is time to try and specify which abstract properties define a query language. This report will concentrate on three important properties:

- **declarative semantics**, meaning that a query has a meaning without the developer having to know how exactly the query gets evaluated
- **closed form evaluation**, which is given when the result of a query can be represented in the language of the database
- **query expressiveness**, is a relative property which defines a query language A to be more expressive than query language B if one can express all queries which are possible in B with A but there are queries one could make in A which aren't possible in B.

4. Applying The Framework To TSQL2 and HRDM

In this chapter the formal framework for temporal databases and query languages will be applied to the temporal query languages TSQL2 and HRDM, so that in the end we can specify the abstract properties of these languages. This chapter takes together the query languages with the corresponding databases, as every specific concrete query language needs a rather specific database to work on.

4.1 TSQL2

4.1.1 A Short Introduction To TSQL2

The TSQL2 specification [7] describes the language as „a temporal extension to the SQL-92 language standard“. This means that new concepts (and keywords) were added to the existing SQL language resulting in TSQL2. So, a „smooth transition from SQL-92 to TSQL2 is possible.“[7]

4.1.2 The Temporal Domain Of TSQL2

The temporal domain of TSQL2 is instant based, although it takes a rather interesting approach by making instants themselves not directly comparable. So, „an instant can only be approximately represented“[7]. This means that it is only possible to compare two instants by specifying with which *granularity* they are to be compared. For example, if we have 2 instant timestamps which both lie in the same day, but in different hours, then we could compare them with the granularity of a day (for which the two instant would be the same) or by the granularity of an hour (for which the instants would be different). Because of this, the temporal domain of TSQL2 does not have to specify whether time is dense or discrete, it isn't important because there are no queries that could give different answers for either of the properties. However, the language specification allows user defined granularities. TSQL2 defines two time axis, valid time and transaction time.

4.1.3 The Abstract Temporal Database Implemented By TSQL2

TSQL2 uses a timestamp view of the underlying database. A timestamp is implicitly associated with each tuple, so the timestamp isn't implemented as just another row in the table. Also, every timestamp records the set of maximal periods in which the corresponding tuple is true. So if in Table 3 the citizen Daniel moves back to Wollerau in 2012 and lives there until 2014, then the timestamp for the row (000-00-0000, Daniel, Wollerau, Student) will have a timestamp that has the initial period from 1995 to 2008 and also the period from 2012 to 2014. The same table may have valid time support, transaction time support as well as both together or no temporal values at all, which consequently will be considered as a snapshot of the data at a given time.

4.1.4 The Abstract Temporal Query Language Of TSQL2

As TSQL2 is an extension of SQL-92, it also shares the declarative semantics of SQL. All queries that are possible in SQL-92 are also possible in TSQL2. One could even argue that TSQL2 is more expressive than SQL-92, although this is a little misleading as TSQL2 relies on an extension of the data model of SQL-92. Without the extended data model it would not be necessary to have the additional semantics for queries. So the expressive power of TSQL2 is essentially the same as that of SQL-92 or any other language based on relational algebra which incorporates support for temporal queries, which also means that it lacks the power to „express recursive queries such as reachability“[8]. However, there have been efforts to get features of TSQL2 into newer SQL standards (which don't have this restriction) in [9]. The language clearly evaluates queries in closed form, as queries return tuples in of datatypes used in the database.

4.2 HRDM

4.2.1 A Short Introduction To HRDM

The Historical Relational Data Model (HRDM) predates TSQL2 by quite a few years. It was defined in the year 1987 in [10]. However, the paper only defines the system using relational algebra and further search did not reveal any concrete, working implementations of a HRDM based system.

4.2.2 The Temporal Domain Of HRDM

HRDM supports an instant based temporal domain with discrete time instants. It only supports one temporal dimension, the granularity of time isn't specified (this probably would be an implementation issue, for example one could set one time instant to correspond to one second.) Generally this is a very simple temporal domain.

4.2.3 The Abstract Temporal Database Implemented By HRDM

HRDM implements a kind of timestamp view based abstract temporal database, where *lifespans* (which correspond to timestamp intervals) can be associated with *objects* (tuples *and* also attributes). As tuples and attributes can have associated lifespans, this means that „tuples are heterogeneous in their temporal dimension“.[10] However, it is important to distinguish this property from multiple temporal dimensions as implemented by other abstract temporal databases. Whereas other databases may allow the *same* tuple to have several temporal values on *different time axes*, HRDM only allows one temporal dimension, but lets the user define lifespans for tuples and attributes separately (consequently having parts of the tuple having two associated temporal values, that from the tuple and that from the attribute.)

4.2.4 The Abstract Temporal Query Language Of HRDM

The query language used is mainly derived from relational algebra by redefining the operators to work in a temporal, so it has declarative syntax (using the relational operators), closed form evaluation, and similar query expressiveness as other languages based on relational algebra. So it shares the restriction of not being able to express recursive queries, but additionally the join and select operators only can compare „attribute values that occur at the same point in time“[11]. Queries that compare snapshots of tables at different times aren't possible, which seems like a quite big restriction.

4.3 Comparison Of the Abstract Properties Of TSQL2 And HRDM

Figures 2 and 3 show small summaries of the information gathered after applying the uniform formal framework to TSQL2 and HRDM. TSQL2 seems more advanced, as it supports a more interesting temporal model than the simple, discreet and one-dimensional model of HRDM. Additionally the expressive power of TSQL2 seems to be clearly greater

then that of HRDM because of HRDM's inability to express queries which compare snapshots of tables at different time instants. The fact that both languages have quite different abstract properties is especially interesting when one wants to know if they are interoperable. Trying to somehow map a TSQL2 query to HRDM would lead to great problems, as we now can easily see after examining their abstract properties. For example, how should one translate a TSQL2 query that compares snapshots at different time instants (which is not possible in HRDM)? Or how should one be able to compare timestamps associated with attributes (which is possible with HRDM) in TSQL2? Having extracted the abstract properties of these languages one can clearly see where exactly the fundamental differences lie.

ABSTRACT PROPERTIES	T S Q L ₂
TEMPORAL DOMAIN	instant based, comparing via granules of time (discreet or dense not important), 2 time dimensions
ABSTRACT TEMPORAL DATABASE	timestamp view, timestamps are set of maximal continuous periods
ABSTRACT TEMPORAL QUERY LANGUAGE	declarative semantics, expressive power similar as for SQL-92 (no recursive queries expressible), closed form evaluation

Figure 2: Summary of some abstract properties of TSQL2

ABSTRACT PROPERTIES	H R D M
TEMPORAL DOMAIN	discreet, instant based, 1 time dimension
ABSTRACT TEMPORAL DATABASE	timestamp view (lifespans on objects), lifespans can be associated with tuples as well as attributes
ABSTRACT TEMPORAL QUERY LANGUAGE	declarative semantics, expressive power similar as for relational algebra (no recursive queries expressible), no comparing of snapshots with different associated timestamps, closed form evaluation

Figure 3: Summary of some abstract properties of HRDM

5. Conclusions And Criticism

After describing the uniform framework for temporal database and applying it to two languages, it seems that the framework does specify the most important properties of a temporal database and its associated query language. However, the paper describing the framework [3] does not do a great job at showing how to apply it on real languages, it is more concerned in describing properties of abstract query languages and their properties. Especially the interoperability between different temporal query languages is only touched very theoretically, mainly describing how it should be possible to map a concrete temporal database to its abstract temporal database and from there back to an other concrete temporal database. How exactly this could be done isn't shown. To be fair, the information that was available at that time about the different concrete temporal query languages was little and has not grown much since that time. So it is rather hard to really compare languages for which one has only got the specification and no real implementation. It also seems that there isn't a so much *recent* research on temporal databases, or, as mentioned in the introduction of this report, the research seems to be mainly concerned with spatiotemporal databases. Efforts made to incorporate more temporal aspects into SQL:1999[9] do not seem to have been very fruitful as the work was given up in 2001. Implemented system seem rare, although there are a few out in the wild, such like the prototype implementation by TimeDB [12] (which uses a similar language to TSQL2) or temporal support in newer versions of the Oracle Workspace Manager [13].

Bibliography

1. Wu, Y., S. Jajoda, and S. Wang, *Temporal Database Bibliography Update*. 1998.
2. <http://sstd09.cs.aau.dk/>.
3. Chomicki, J., *Temporal Query Languages: a Survey*. 1995.
4. Tsichritzis, D. and F. Lochovsky, *Data Base Management Systems*. 1977.
5. Bettini, C., et al., *A glossary of time granularity concepts*. 1998.
6. Jensen, C., et al., *A Consensus Glossary Of Temporal Database Concepts*. 1998.
7. Snodgrass, R.T., et al., *TSQL2 Language Specification*. 1994.
8. Libkin, L., *Expressive Power of SQL*. 2001.
9. Snodgrass, R., et al., *Transitioning temporal support in TSQL2 to SQL3*. 1998.
10. Clifford, J. and A. Croker, *The Historical Relational Data Model (HRDM) And Algebra Based On Lifespans*. 1987.
11. Clifford, J., A. Croker, and A. Tuzhilin, *On Completeness Of Historical Relational Query Languages*. 1994.
12. <http://www.timeconsult.com/Software/Software.html>.
13. www.oracle.com.