Author(s): J.E.P. Wijnands; W.L.A. Derks Date: January 1999

Temporal databases

State-of-the-art

KPN Research

Informationsheet issued with Report 30432

Title:	Temporal databases: State-of-the-art
Abstract:	To investigate the added value of temporal databases, the *Temporal Database project* has been started within the *IT Strategies* program. This report is the result of the first activity of this project which consisted of a literature study in the area of temporal databases. It describes the state-of-the-art of temporal databases from an academic point of view as well as from a commercial point of view.
Author(s):	J.E.P. Wijnands; W.L.A. Derks
Reviewer(s):	W. Jonker; M.H. Böhlen (Aalborg University)
Key Words:	time modelling, database, history analysis, temporal database, time series

KPN Research
PO box 15000
9700 CD Groningen
The Netherlands

email: {J.E.P.Wijnands, W.L.A.Derks} @kpn.com

Contents

Management Summary

In our information society, the amount of data needed for business decisions, is increasing at an enormous speed. Not only data of the current situation is stored, but more and more data related to historical events is stored. In a telecommunication context, examples are historical information on priceplans of customers for the past three years or occurred network errors for the past six months. This historical data could be used for prediction of customer or network behaviour. A more general example of systems containing time are data warehouses. The number of data warehouse databases for analysis purposes is increasing. Currently, the emphasis is on support for marketing, sales and management but e.g. ICT management could also be supported by data warehouses.
Current applications only make limited use of historical data, because it is rather complex to model and reason over time. For applications to make better use of historical data, implicit modelling and support of time in the database is desirable. Databases providing this kind of support are called *temporal databases*.

To investigate the added value of temporal databases, the *Temporal Database project* has been started within the *IT Strategies program*. This report is the result of a literature study in the area of temporal databases. It describes the state-of-the-art of temporal databases from an academic point of view as well as from a commercial point of view. The report gives a detailed description of the concepts behind modelling and reasoning with time and their application in commercial products. As such, this report is intended for technical readers.

After some twenty years of temporal database research, consensus on concepts and aspects has been reached within the temporal database community. The concepts of Valid Time (*when is a fact true in reality*) and Transaction Time (*when is a fact stored in the database*) are considered the cornerstones for temporal support. As compatibility with existing database systems has high priority, most researchers have tried to extend the relational or the object oriented model. Extensions of the relational model where the most succesful probably because most current databases are relational databases. The ideas and concepts of the ATSQL2 project are the most likely candidates to be incorporated in the temporal module of the new SQL3 standard as they are fully upward compatible with existing relational database applications and provide easy to use, powerful SQL expressions for reasoning over historical data.

Up to now, major database vendors like Oracle, Informix, IBM and Microsoft do not show real interest for temporal support in their products althought prototypes and commercial front-ends have shown the possiblities of it. When asked for it, the vendors refer to their support of time series. But time series databases are not the same as temporal. We have the feeling that database vendors and database users are waiting for each other. As long as vendors do not offer temporal support, the customers are not aware of the potentials of it and as long as the customers do not ask for temporal support, the vendors will not offer it. For the time being, comparing experiments should be carried out to get a better idea of the added value of temporal support.

Glossary

taken from [TDBG98]

Name	Definition
Bitemporal Relation	A bitemporal relation is a relation with exactly one system-supported valid time and exactly one system-supported transaction time. As with valid-time relations and transaction-time relations, there are no restrictions concerning how either of these temporal dimensions may be incorporated into the tuples.
Calendar	A calendar provides a human interpretation of time. As such, calendars ascribe meaning to temporal values where the particular meaning or interpretation is relevant to the user. In particular, calendars determine the mapping between human-meaningful time values and an underlying time-line.
Chronon	In a data model, a one-dimensional chronon is a non-decomposable time interval of some fixed, minimal duration. An n-dimensional chronon is a non-decomposable region in n-dimensional time. Important special types of chronons include valid-time, transaction-time, and bitemporal chronons.
Coalescing	The coalesce operation takes as argument a set of value-equivalent tuples and returns a single tuple which is snapshot equivalent with the argument set of tuples.
Duration	A duration is an non-directed amount of time with known length, but no specific starting or ending instants. For example, the duration ‘‘one week” is known to have a length of seven days, but can refer to any block of seven consecutive days.
Dynamic Valid-time Partitioning	In dynamic valid-time partitioning the valid-time elements used in the partitioning are determined solely from the timestamps of the relation.
Event	An event is an instantaneous fact, i.e., something occurring at an instant. An event is said to occur at a chronon t if it occurs at any instant during t.
Instant	An instant is a time point on an underlying time axis.
Instantaneous Aggregation	In instantaneous aggregation, for each chronon on the valid time-line, the aggregate is applied to all tuples valid at that instant.
Lifespan	The lifespan of a database object is the time over which it is defined. The valid-time lifespan of a database object refers to the time when the corresponding object exists in the modeled reality. Analogously, the transaction-time lifespan refers to the time when the database object is current in the database.
Macro-event	A macro-event is a wholistic fact with duration, i.e., something occurring over an interval taken as a whole. A macro-event is said to occur over an interval I if it occurs over the set of contiguous chronons representing I (considered as a whole).
Schema Evolution	A database system supports schema evolution if it permits modification of the database schema without the loss of extant data. No support for previous schemas is required.
Schema Versioning	A database system accommodates schema versioning if it allows the querying of all data, both retrospectively and prospectively, through user-definable version interfaces.
Snapshot Relation	Relations of a conventional relational database system incorporating neither valid-time nor transaction-time timestamps are snapshot relations.
Span	A span is a directed duration of time. A span is either positive, denoting forward motion of time, or negative, denoting backwards motion in time.
Static Valid-time	In static valid-time partitioning the valid-time elements used are determined

Partitioning	solely from fixed points on a calendar, such as the start of each year.
Temporal Data Type	The user-defined temporal data type is a time representation specially designed to meet the specific needs of the user. For example, the designers of a database used for class scheduling in a school might be based on a ‘‘Year:Term:Day:Period” format. Terms belonging to a user-defined temporal data type get the same query language support as do terms belonging to built-in temporal data types such as the DATE data type.
Temporal Database	A temporal database is a database that supports some aspect of time, not counting user-defined time.
Temporal Element	A temporal element is a finite union of n-dimensional time intervals. Special cases of temporal elements include valid-time elements, transaction-time elements, and bitemporal elements. They are finite unions of valid-time intervals, transaction-time intervals, and bitemporal intervals, respectively.
Temporal Modality	Temporal modality concerns the way according to which a fact originally associated with a chronon or interval at a given granularity distributes itself over the corresponding chronons at finer granularities or within the interval at the same level of granularity.
Time Interval	A time interval is an anchored span. In a system that supports a time line composed of chronons, an interval may be represented by a set of contiguous chronons.
Timestamp	A timestamp is a time value associated with some object, e.g., an attribute value or a tuple. The concept may be specialized to valid timestamp, transaction timestamp, interval timestamp, instant timestamp, bitemporal-element timestamp, etc.
Transaction Time	A database fact is stored in a database at some point in time, and after it is stored, it is current until logically deleted. The transaction time of a database fact is the time when the fact is current in the database and may be retrieved. Transaction times are consistent with the serialization order of the transactions. Transaction-time values cannot be later than the current transaction time. Also, as it is impossible to change the past, transaction times cannot be changed. Transaction times may be implemented using transaction commit times, and are system-generated and -supplied.
Transaction-time Relation	A transaction-time relation is a relation with exactly one system supported transaction time. As for valid-time relations, there are no restrictions as to how transaction times may be incorporated into the tuples.
User-defined Time	User-defined time is an uninterpreted attribute domain of date and time. User-defined time is parallel to domains such as ‘‘money” and integer - unlike transaction time and valid time, it has no special query language support. It may be used for attributes such as ‘‘birth day” and ‘‘hiring date.”
Valid time	The valid time of a fact is the time when the fact is true in the modeled reality. A fact may have associated any number of instants and time intervals, with single instants and intervals being important special cases. Valid times are usually supplied by the user.
Valid-time Cumulative Aggregation	In cumulative aggregation, for each valid-time element of the valid-time partitioning (produced by either dynamic or static valid-time partitioning), the aggregate is applied to all tuples associated with that valid-time element. The value of the aggregate at any instant is the value computed over the partitioning element that contains that instant.
Valid-time Partitioning	Valid-time partitioning is the partitioning (in the mathematical sense) of the valid time-line into valid-time elements. For each valid-time element, we associate an interval of the valid time-line on which a cumulative aggregate may then be applied.
Valid-time Relation	A valid-time relation is a relation with exactly one system supported valid time. There are no restrictions on how valid times may be incorporated into the tuples; e.g., the valid-times may be incorporated by including one or more additional valid-time attributes in the relation schema, or by including the valid-times as a component of the values of the application-specific attributes.

List of Abbreviations

DBMS	DataBase Management System
EER	Extended Entity Relation (model)
GIS	Geographic Information System
ICT	Information and Communication Technology
NONSEQ	Non-Sequenced
JDBC	Java Data Base Connectivity
MCDB	Mobile Customer care DataBase
ORDBMS	Object Relational DBMS
SEQ	Sequenced
SQL	Structured Query Language
TDB	Temporal DataBase
TDBMS	Temporal DBMS
TDT	Terrestrial Dynamic Time
TSMS	Time Series Management System
TSQL	Temporal SQL
TUC	Temporal Upward Compatible
UC	Upward Compatible
UTC	Universal Time Coordinated

1 Introduction

More and more, organisations try to collect as much as possible data on customers, competitors, products, services, processes etc. to be able to better meet the needs of their customers. Inherently, historical data will be part of the total collection of information. In a telecommunication context, examples are historical information on priceplans of customers or occurred network errors. An important category of systems containing time related data are the data warehouse systems. The number of data warehouse databases for analysis purposes is increasing. Currently, the emphasis is on support for marketing, sales and management but e.g. ICT management could also be supported by data warehouses. In these data warehouses, time is an important aspect. Besides data on the current situation, data on the history is stored. Current applications only make limited use of historical data, because it is rather complex to model and reason over time. For applications to make better use of historical data, implicit modelling and support of time in the database would be desirable. Databases providing this kind of support are called *temporal databases*.

Temporal queries are said to be shorter and easier to formulate resulting in improved productivity, correctness and maintainability of applications. Currently, time is explicitely modelled by the database designer and rather complex application code takes care of reasoning over time. Putting temporal support in the database instead of in the application would increase the degree of independence between data and application and would make temporal support available for all applications without the need to reinvent the wheel.

To investigate the added value of temporal databases, the *Temporal Database project* has been started within the *IT Strategies program*. This report is the result of a literature study in the area of temporal databases. It describes the state-of-the-art of temporal databases as well from an academic point of view as from a commercial point of view. The report gives a detailed description of the concepts behind modelling and reasoning with time and their application in commercial products. As such, this report is intended for technical readers. Chapter 2 explains the concepts of temporal databases and temporal reasoning. Chapter 3 then continues with the description of a proposed temporal query language for the new SQL3 standaard. Finally, chapter 4 describes existing commercial and prototype temporal databases.

2 Scientific

This chapter describes the scientific state-of-the-art of temporal databases. After an introduction of concepts of time related issues, modeling of time and reasoning over time is discussed. The chapter ends with a description of some temporal database related topics.

2.1 Motivation

To illustrate the added value of a database having notion of time, let's start with a small example of a customer care database (adapted from [ZCFS+97]). The database contains all kinds of data on customers for marketing and management purposes. One of the tables in the database contains customer priceplan and turnover-category data. In a relational database, this can be represented as follows:

```
Customer( Name, Priceplan, Turnover-cat)
```

In the most simple case, we only record the current priceplan and turnover-category for all customers. With a simple SQL select-statement we can retrieve customer data from this table:

```
Select *
From Customer
Where Name = "Temp DB Inc."

Name         Priceplan    Turnover-cat
-------------------------------------
Temp Db Inc. RoyalPlan    Medium
```

For marketing and management purposes, this is not enough. We want to know more about the history of customers. In a regular relational database, we have to explicit add two date attributes to indicate when the information became valid and when the information was not valid anymore.

```
Customer( Name, Priceplan, Turnover-category, Startdate, Enddate)
```

When extracting customer information from the database, the result could look like the following:

```
Name         Priceplan    Turnover-cat Startdate    EndDate
-------------------------------------------------------------
Temp Db Inc. BudgetPlan   Low          01-01-1998   01-05-1998
Temp Db Inc. BudgetPlan   Medium       02-05-1998   12-07-1998
Temp Db Inc. EasyPlan     Medium       13-07-1998   01-12-1998
Temp Db Inc. RoyalPlan    High         02-12-1998   99-99-9999
```

This customer started as a small (less important) one and became an important customer within a year. The 99-99-999 date indicates that this information is valid *until changed*. Adding attributes to record valid dates is not much of a work.
For querying, the difference between a database system with or without notion of time is much more significant.

If we want to know the current situation, the following (more complex) SQL query is needed:

```
Select *
From Customer
Where Name = “Temp DB Inc.”
 and Startdate <= CURRENT_DATE and Stopdate >= CURRENT_DATE
```

It becomes even more complex when we ask the legitimate question about the maximun period of time for which a customer has a certain priceplan. The result should be:

```
Name         Priceplan    Startdate    EndDate
-----------------------------------------------
Temp Db Inc. BudgetPlan   01-01-1998   12-07-1998
Temp Db Inc. EasyPlan     13-07-1998   01-12-1998
Temp Db Inc. RoyalPlan    02-12-1998   99-99-9999
```

To calculate this result, several valid periods for each priceplan have to be *coalesced* i.e. overlappping and adjacent periods have to be joined. This requires the following complex SQL statements for making a temporary table containing the data to answer the question:

```
CREATE TABLE Temp(Name, Priceplan, Start, Stop)
AS     SELECT Name, Priceplan, Startdate, EndDate
       FROM Customer
       WHERE Name = “Temp Db Inc.”;

SELECT DISTINCT F.Start, L.Stop
FROM Temp AS F, Temp AS L
WHERE F.Start < L.Stop
       AND F.Priceplan = L.Priceplan
       AND NOT EXISTS (SELECT *
              FROM Temp AS M
              WHERE M.Priceplan = F.Priceplan
              AND F.Start < M.Start AND M.Start < L.Stop
              AND NOT EXISTS (SELECT *
                     FROM Temp AS T1
                     WHERE T1.Pricplan = F.Priceplan
                     AND T1.Start < M.Start AND M.Start <= T1.Stop))
       AND NOT EXISTS (SELECT *
              FROM Temp AS T2
              WHERE T2.Priceplan = F.Priceplan
              AND ((T2.Start < F.Start AND F.Start <= T2.Stop) OR
              (T2.Start < L.Stop AND L.Stop < T2.Stop)));
```

The equivalent query in a temporal SQL would be something like (see also chapter 3.7):

```
(sequenced valid select name, priceplan from customer)(vt);
```

The above examples illustrate the increasing complexity of application code, even for simple questions, when the database has no notion of time.

Before we proceed, we first introduce the fundamental concepts of time in temporal database research.

2.2 Time semantics

Time is represented by a *timeline.* Any point on the timeline is called a *time instant* which has no duration. The time continuum between two time instants is called a time span and is directed (i.e. it does either direct into the future or into the past). Because both measurements in real world have restricted accuracy and the numerical accuracy of computer systems is bounded, the timeline is partitioned by undirected time units with fixed duration. Such a fixed time unit is called a *chronon*. Chronons are the smallest, nondecomposable time units associated with time and they partition the time line. Adjacent chronons make up a *time interval*. A time interval is directed as well as a time span. A time interval is thus a time span which is bounded by chronons. The union of multiple time intervals makes up a *temporal element*.

See Figure 2.1 for a graphical representation of the definitions.

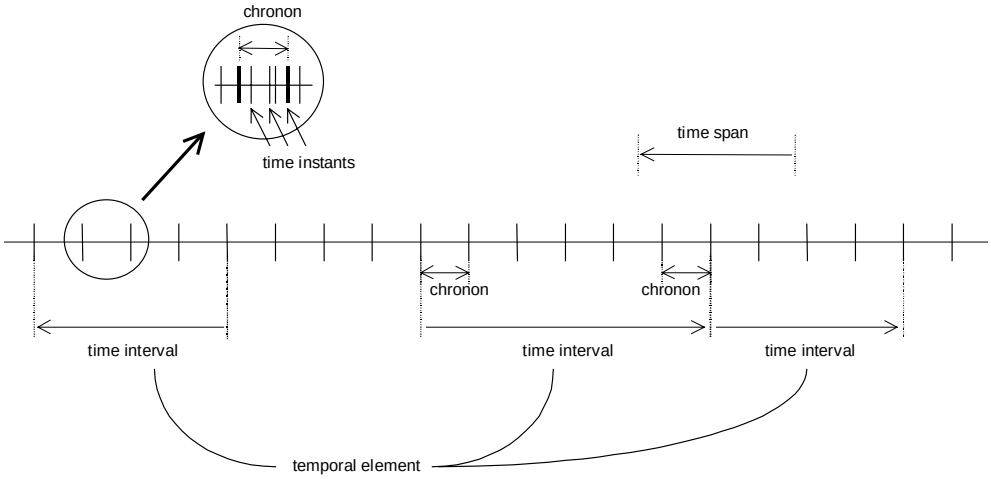


Figure 2.1: Graphical representation of time concepts

2.2.1 Time domains

To be able to reason about time distance and to provide for an unambiguous interpretation of a time instant, the time domain is associated with a mathematical domain. The time domain can be viewed as *discrete* or *continuous*. A discrete time model is isomorphic to the natural numbers, which implies that each time instant has a unique successor. A continuous model can either be *dense* or not. A dense continuous model is isomorphic with the real numbers, which means that there are no 'gaps' on the time line. This is unlike the rational numbers, which are called *not dense.*

Time domain	**discrete**	**continuous**
dense	N.A.	real numbers
not dense	natural numbers	e.g. rational numbers

The discrete time domain is generally used (by using chronons), because time measures are inherenty imprecise. In addition one tends to reason in terms of discrete time units. For example, time is often denoted in terms of seconds, milliseconds or minutes. Maybe the most important reason for not adopting continuous domains is because continuous domains are more difficult to implement than discrete domains.

2.2.2 Models of time

In the previous paragraph we have introduced the notion of time line. Although it appears to be straightforward to have only linear time line, this is not the case . Three time models exist:

- linear time,
- branching time,
- circular time.

Linear time is the traditional look on time, in which all time instants are linearly ordered. *Branching time* is used when several alternatives have to be considered (e.g. possible futures). This may be the case in planning systems where multiple scenario's have to be incorporated. Time points are only partially ordered in branching time. *Circular time* can be used to represent recurring events. One can think of a period of a year in which the same events occur at the same date (e.g. birthdays) or the days of a week. Calendars are most often cyclic (e.g. Gregorian calendar, in which days and months cycle).

2.2.3 Time dimensions

In the previous paragraphs we have defined time and its related concepts. Now we define what kinds of time we want to reason about. There are two dimensions of time in a temporal database that may have explicit temporal support:

1. transaction time,
2. valid time.

Transaction time represents the time a fact is stored in the database, i.e. models the time behaviour of an object in the database. Because transaction time is registered immediately, the time stamps are consistent with the serialization order of the transactions. Transaction times are system generated and cannot be changed.
Valid time is the time a stored fact is valid in reality. The valid time can therefore be in the past, present or future. The valid time of a fact is determined by the user and can be changed afterwards.

In addition to transaction time and valid time there is *user-defined* time. User-defined time is an attribute domain for specifying time points. Typical examples are 'birthday' and 'hiring date'. This time domain is not treated differently from other attribute domains apart from the fact that some special time functions are supported. Examples of these functions are 'MONTHS_BETWEEN' (to determine the number of months between two dates) or type conversion functions like 'TO_DATE()' or 'YEAR()'. User-defined time is not treated differently from any other attribute domain by the query optimizer. Therefore we do not consider user-defined time as a time dimension in this report.

To illustrate the use of transaction time and valid time, consider the following example. At 15 May 1998 the fact that our Customer *Temp DB* had a priceplan *BudgetPlan* from 1 January 1998 until 01 May 1998, is stored in a temporal database. This would then be stored as:

```
Name      PricePlan   Valid time              Transaction time
-------------------------------------------------------------
Temp DB  BudgetPlan  01-Jan-1998/01-May-1998  15-May-1998/until changed
```

According to the support of the two time dimensions defined above, we can identify four types of databases based on temporal support:

1. snapshot database,
2. valid time database,
3. transaction time database,
4. bitemporal database.

A *snapshot database* has neither valid time support nor transaction time support. Only present knowledge about the objects is stored. There is no DBMS support for neither reasoning about nor storing time information for the objects. Hence, complex queries must be formulated to retrieve temporal data. Storage of temporal data may only be achieved by using user-defined time attributes. Most current database systems are snapshot databases. Snapshot databases are used when only the current status of objects is relevant.

A *valid time database*[1] has support for valid time reasoning and storage. The objects stored in the database may have a valid time associated with them. Valid time databases are used when reasoning about history and future information is required, but only current knowledge about the objects is relevant. An example is someone's agenda. The appointments take place somewhere in time, but it is irrelevant when these appointments have been written in the agenda.

[1] Also called *historical database*

A *transaction time database*[2] has support for transaction time. For every new transaction on the database a new record is created. When an object is updated, the old object is logically deleted (i.e. is marked as being deleted) and a new object is added to the database as the current object. A transaction time database is used when it is relevant how the current status of the stored objects has been reached. For example, for stock control not only the current level is interesting, but also the *process* of how this stock level has been reached.

A *bitemporal database* supports both valid time and transaction time.

Temporal support may exist at different levels: attribute, tuple and schema (see paragraph 2.3.2.1)Temporal support at attribute level means that each change in the state of the attribute is annotated with temporal information. In case of valid time support at attribute level, this means that the attribute value is accompanied with its valid time period. The same goes for tuple level support, where one tuple (with multiple attribute values) is associated with one time period. Temporal support at schema level is defined as schema versioning. Each time changes are made to the schema meta-data, the old state is retained with the appropriate transaction time period. *Schema versioning* should not be confused with *schema evolution*, because the latter has no transaction time support, i.e. cannot revert to previous versions (see glossary).

A special form of temporal database is a *degenerate database*. This is a bitemporal database where the valid and transaction times of a fact are identical. An example of an application is a process monitoring system, where all events are immediately written to the database and hence the database's content is always up-to-date. In that case the real-life state time periods of the process (valid time) is identical to the storage time periods (transaction time).

2.2.4 Storage structures

The storage structure of temporal databases is different from conventional (snapshot) databases. To clearify the difference, we look at the storage of a relation in the four different databases. The examples are taken from [SA86].

2.2.4.1 Snapshot database

Figure 2.2 shows five tuples in a snapshot database. As no transaction time nor valid time is supported, the structure is straightforward.

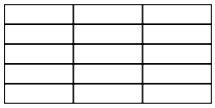


Figure 2.2 One relation in a snapshot database

2.2.4.2 Transaction time database

In a transaction time database the flat structure of a snapshot database is extended with the transaction time dimension. Figure 2.3 shows three states of the database.

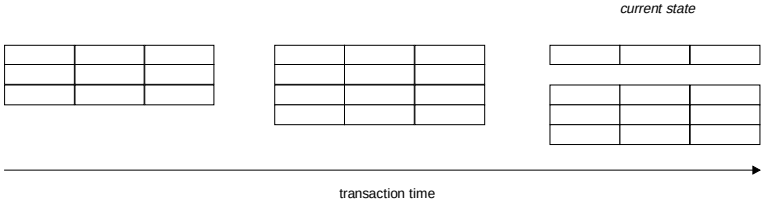


Figure 2.3 One relation in a transaction database

[2] Also called *rollback database*

Initially, the relation contains three tuples. Then one tuple is inserted and the database goes into the second transaction time state. After that, the second tuple is deleted and another tuple is inserted. Remember that all three database states remain in the database. One can ask for data from the database at a certain transaction time. After three states, eleven tuples are logically present in the database. Note that this does not mean, that all eleven tuples necessarily have to be physically present in the database. A more efficient storage strategy may be used to limit the required storage space

2.2.4.3 Valid time database

A valid time database provides only the current state of the relation and has no support for reverting to previous states of the relation. Each tuple has an associated valid time indicating when the fact is valid in the real world. For each tuple in the relation a history can be kept but one cannot see when, i.e. at which transaction time a certain fact was stored in the database. Figure 2.4 shows a valid time relation.

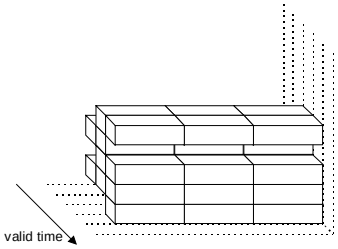


Figure 2.4 One valid time relation

The uppermost tuple in the relation has only values during two valid time intervals. The lowermost tuple is only defined during one valid time interval. Note that all tuples are visible for the user all the time.

2.2.4.4 Bitemporal database

The bitemporal database combines both dimensions, which results in the picture as shown in Figure 2.5.

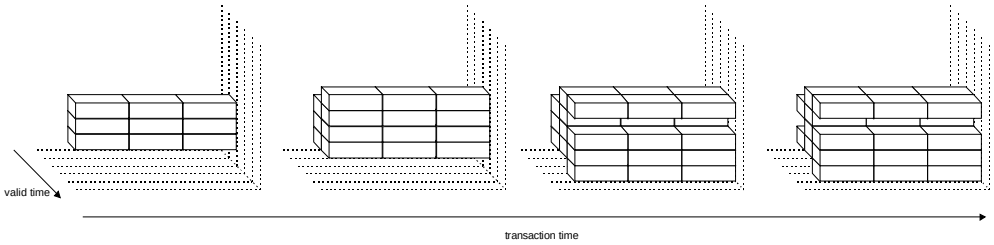


Figure 2.5 One relation in a bitemporal database

Initially, the relation contains three tuples which are valid at the same interval in valid time. In the next transaction, a new tuple (the uppermost) is added which is valid in the next valid time interval. Furthermore, we assure that the three existing tuples are valid in this new valid time period as well. In the second transaction, again a tuple is added (the lowermost) that is valid in the new valid time interval. Three of the already existing tuples are also valid in this new valid time interval, but one tuple is not longer valid in this valid time interval. In the final transaction, the valid time of the second tuple from the first transaction is changed. The valid times of the other tuples do not change and there are no new tuples added.

2.3 Data models

Now we have introduced time concepts, we want to capture these concepts into a data model. This way we can capture the semantics concisely, coherently and consistently. Many data models have been proposed. Most of the propositions are extensions of existing models because this is generally seen as the most promising approach ([OS95]). Time has been added to:

- entity-relationship model,
- knowledge-based data models,
- deductive data models,
- relational models,
- object-oriented models.

Most temporal databases are based on the relational and object-oriented models. Therefore we will address only those two models.

Temporal data models differ in support on four dimensions:

- time representation,
- level of temporal support,
- temporal dimension support,
- base data model.

The time representation is the way time is modelled, i.e. as points or intervals. The level of temporal support can be at attribute-, tuple- and schema level. The temporal dimension support can be a combination of transaction time and valid time. The base data model is one of the data models mentioned above.

In the next paragraphs the four dimensions are discussed further.

2.3.1 Time representation

Time can be represented in two ways:

- points,
- spans.

When time is represented by points, a single time value is associated with a database fact. Time representation by spans is done by defining a start time point and an end time point (i.e. a time span)[3].

Whether either points or spans should be adopted depends on the process to be modeled. In case of discrete processes (events) spans are appropriate and in case of continuous processes points are more appropriate. Spans can be used for discrete processes, because discrete processes have a stable state during some period of time. Examples of discrete processes are temporal relationships, database states or more practical: the state of a traffic light. Continuous processes cannot be modeled accurately by spans, because their state changes continuously. Examples of continuous processes are pressure measurements in chemical processes or the temperature of the weather.

When samples of continuous processes are stored, this is called *explicit modeling*. Explicit modeling can become a problem when values are asked which are not stored. A solution is then to store enough samples and take the most similar time stamp to obtain satisfying accuracy. However, this may require too much storage. Then interpolation function can be used to estimate intermediate values. When functions are stored instead of samples this is called *implicit representation*.

[3] Note that spans can model points as well.

In the time modeling theory, spans have the disadvantage of not being closed under set theoretic operations (e.g. union, intersection, difference). Performing an operation on elements (viz. spans) from the set of spans could result in an set of spans, which is not a single span thus not element of the set of spans. To overcome this problem, combinations of these intervals are supported by the data model; these sets of intervals are called *temporal elements or* elementary subsets.

2.3.2 Levels of temporal support

There are two types of temporal support. There is support for:

- data,
- meta data.

The data temporal support is the most obvious and supports valid and transaction time for stored database facts. The meta data temporal support is defined at meta data level (i.e. schema level). Both types will be discussed in the next paragraphs.

2.3.2.1 Data level

Temporal support at data level can be defined at two levels of granularity:

- attribute level,
- tuple level.

When time is supported at attribute level, time can be associated with each attribute value of the relation/object class. When time is supported at tuple level, time is associated with the whole instance. Some models also support groups of attributes within one tuple. However, most models either support attribute or tuple level.Table 2.1 and Table 2.2 give examples of attribute and tuple level time support respectively.

Name(Startdate; EndDate)	Priceplan (Startdate; EndDate)
Temp Db Inc. (01-01-1998; 99-99-9999)	BudgetPlan (01-01-1998; 12-07-1998)
	EasyPlan (13-07-1998; 01-12-1998)
	RoyalPlan (02-12-1998; 99-99-9999)

Table 2.1 Attribute time stamping

Name	Priceplan	(Startdate; EndDate)
Temp Db Inc.	BudgetPlan	(01-01-1998; 12-07-1998)
Temp Db Inc.	EasyPlan	(13-07-1998; 01-12-1998)
Temp Db Inc.	RoyalPlan	(02-12-1998; 99-99-9999)

Table 2.2 Tuple time stamping

The advantage of attribute time support is that this is an accurate way of describing the temporal behavior of an attribute. When the value of only one attribute changes, the new value is stored with the corresponding temporal information. This is different for tuple time stamping. In that case all static attributes are duplicated and stored into a new tuple. This introduces redundancy and hence wastes storage space. However, the widely accepted relational model can be extended in a straightforward way to tuple time stamping and hence tuple time stamping is adopted most frequently in the relational model.
The attribute and tuple time stamping are also called grouped and ungrouped respectively. In addition the term heterogeneous and homogeneous are used as well.

Another discussion about time modelling is the matter of single- or multivalued time fields. In single valued time fields only one time representation is present (i.e. one point

or one span) at each time level (i.e. a database fact is associated with a single point or a single span). In multivalued time fields multiple points or spans may be combined in one time entry[4] (i.e. one database fact is associated with multiple time points or for example a union of multiple time spans). In our opinion, this discussion is very similar to the field of normalization. Multi- or single-valued time models are similar to the non-first normal form and first normal form. [Dey96] describes the mapping from the conventional relational model to the temporal relational model in detail and defines four temporal normal forms.

2.3.2.2 Meta data level

Temporal support at meta data level translates into time support for schema's. Meta data time support enables reasoning about different versions of the schema and hence makes it possible to revert to past states of the model. This functionality is useful to be able to reason about data that was differently structured in the past. Especially when legacy systems are replaced, the schema is likely to be migrated to the current system. When old data should be retrieved, the old schema is required.

Two important terms are often used in this context: *versioning* and *evolution*. Both terms describe the migration from one schema to another, with the difference that versioning retains past states (history) and evolution only migrates, but does not store previous schema's.

Note that schema's persist in one state for some time span. This makes schema versioning a discrete process. Therefore schema versioning can be modeled by points and spans.

2.3.3 Base models

The data models are mostly based on the relational model. In addition other models are extended with time support, among which the (extended) entity-relationship model (EER) is one.[5] The relational model is extended with temporal values that are associated either with tuples or attributes. However, accurate modeling of time may become complex and may require non-first-normal forms. Some researchers think that the relational model is too simplistic to model time accurately. They feel it more naturally to extend the object-oriented models with time concepts.

Appendix B gives a table with temporally extended relational and object-oriented models.

2.3.4 Temporal dimensions

The two time dimensions valid time and transaction time describe different processes and hence may have different representation. Valid time can model discrete and continuous processes, whereas transaction time models database states only, which is a discrete process.

For valid time, point representation can be used to describe samples of a continuous process or for description of (discrete) events. Span representation is used for discrete processes. In case of spans, duration of a state is modeled explicitly. When this duration is infinite into the future, a value called 'forever' is used to indicate the infinity of the span.

---

[4] Also called *temporal elements*.

[5] The temporal extension is called TEER and is described in [TCGJ+93].

[ZCFS+97] gives a table of data models that have appeared in literature.

	Point[6]	Span[7]	Set of spans[8]
Timestamped attribute values	ADM Caruso Lorentzos	Bassiouni Gadia-2 McKenzie Tansel	Bhargava Gadia-1 HRDM TOODM
Timestamped groups of attributes	Sciore-2		
Timestamped tuples	Ariav EDM HDM Lum Sadeghi Segev Wiederhold	Ahn Ben-Zvi Jones Navathe Sarda Snodgrass Yau	BCDM
Timestamped objects	TEDM	OSAM*/T TMAD	

Table 2.3 Valid time in temporal data models (taken from [ZCFS+97])

As was mentioned before, transaction time models the discrete process of database state changes. Therefore, the transaction time cannot be specified by the user. The process can be represented by points or spans (see paragraph 2.3.1). In case the time is represented by spans, the current state of a database fact is bounded by it's starting time point and ends with a special value. This value may be 'now', 'forever' or 'until changed'. When the state of the database fact changes (i.e. due to a transaction), this special value may be replaced by the current time value.

[ZCFS+97] gives a table of data models that support transaction time.

	Point[9]	Span[10]	Three points[11]	Set of spans[12]	Other
Timestamped attribute values	Caruso			Bhargava TOODM	Sciore-1
Timestamped groups of attributes	Sciore-2				OVM
Timestamped tuples	Ariav DATA DM/T EDM Lomet	Postgres Snodgrass Yau	Ben-Zvi	BCDM	
Timestamped objects	IRIS TIGUKAT				IRIS Kim
Timestamped sets of tuples	ADM Ahn	McKenzie			
Object graph	MATISSE TIGUKAT				MATISSE
Timestamped schema	MATISSE TIGUKAT	McKenzie Postgres		BCDM	MATISSE

Table 2.4 Transaction time in temporal data models (taken from [ZCFS+97])

6 In the original table this was 'Single chronon'.

7 In the original table this was 'Period (pair of chronons)'.

8 In the original table this was 'Valid-time element (set of periods)'.

9 In the original table this was 'Single chronon'.

10 In the original table this was 'Period (pair of chronons)'.

11 In the original table this was 'Three points'.

12 In the original table this was 'Valid-time element (set of periods)'.

The data model of Ben-Zvi is a special case, because it uses three points to represent the time information of a fact. Ben-Zvi's model records:

1. the transaction time when the valid start time was recorded,
2. the transaction time when the valid stop time was recorded,
3. the transaction time when the tuple was logically deleted.

2.4 Reasoning

In the previous paragraphs we have identified the aspects of time modeling. In this paragraph we describe issues concerning the reasoning about time. We restrict to issues that are different from the traditional reasoning. Five issues are typical:

- surrogates,
- coalescing,
- granularity,
- aggregates,
- vacuuming.

Each of these issues is described below.

2.4.1 Surrogates

In temporal databases it is possible that the primary key of an object is time-varying. To be able to identify these objects through time, *surrogates* are introduced. These are unique values that can be compared for equality but are otherwise not visible to users.

2.4.2 Coalescing

An important characteristic of temporal database management systems is the capability of restructuring the results of a temporal query. When the same values are valid over some period of time, these time periods are merged and presented as one period. This restructuring principle is called *coalescing*.

Look at the following results as an example. It is the intermediate result of a query which retrieves all price plan contracts which were active during 1998. No coalescing has been done so far.

```
PricePlan    Valid_time
-----------------------------------
BudgetPlan   [01-01-1998; 17-01-1998]
BudgetPlan   [01-01-1998; 08-02-1998]
BudgetPlan   [01-01-1998; 26-02-1998]
EasyPlan     [18-01-1998; 21-02-1998]
RoyalPlan    [22-01-1998; 31-12-1998]
EasyPlan     [09-02-1998; 12-03-1998]
EasyPlan     [27-02-1998; 08-03-1998]
BudgetPlan   [09-03-1998; 31-12-1998]
RoyalPlan    [13-03-1998; 31-12-1998]
```

After coalescing the active periods of the different price plans are merged together as far as possible. The result looks like:

```
PricePlan    Valid_time
-----------------------------------
BudgetPlan   [01-01-1998; 26-02-1998]
BudgetPlan   [09-03-1998; 31-12-1998]
EasyPlan     [18-01-1998; 12-03-1998]
RoyalPlan    [22-01-1998; 31-12-1998]
```

Apparently, the BudgetPlan value is present over the periods 01-01-1998 until 26-02-1998 and from 09-03-1998 until 31-12-1998. The EasyPlan value is valid from 18-01-1998 until 12-03-1998 and the RoyalPlan is valid from 22-01-1998 until 31-12-1998.

2.4.3 Granularity

To be able to reason over time, some agreement about the time unit must be made. This level at which is reasoned over time is called *temporal granularity*. Examples of different temporal granularity are seconds, days or months. A second is defined at a *finer temporal granularity* than a day. A year has coarser granularity than a month. What temporal granularity levels are known in the database depends on the *calendar* used. The calendar describes the partitioning of the time line into time units and defines the granularity of each unit. In addition the calendar also defines the mapping between the time units. The finest granularity that can be defined is the granularity of a chronon, which is system defined. Note that the calendar may have a coarser finest level than the chronon.

Reasoning about facts at different granularities introduces *temporal modalities*. For example, a customer has subscribed to a service during some days in May and September 1998. The question 'Was the customer subscribed at January, 1st, 1998?' answers to a sure 'no'. The same question for May 13th, 1998 will result in a 'maybe'. However, the question whether the customer was subscribed during May 1998 results in a 'yes'. The 'maybe' result is a *indeterminate* answer for a question at day level, but the answer gets determinate when the granularity is raised to month level.

Operands must be the same granularity in order to be comparible. To accomplish this, conversion functions exist. CAST and SCALE are such functions in TSQL2.

2.4.4 Aggregates

A temporal aggregate is an aggregate that returns time-varying results when applied to temporal relations. Calculating temporal aggregates is a two step process:

1. partitioning of time line,
2. computing aggregates.

Two partitioning strategies are possible: static or dynamic. Static partitioning divides the time line in predefined pieces (e.g. days or months). Dynamic partitioning determines the pieces of the time line by grouping the relevant values of the set and then coalesce this result. This way the partitioning is created dynamically. After the timeline has been partitioned, per time slice the aggregate is calculated.

Besides the traditional aggregate operators MIN, MAX, COUNT, SUM and AVG a new aggregate operators have been defined in various temporal languages. In TSQL2 the aggregate operator RISING is introduced. This operator evaluates to the longest period during which a specified attribute value was monotonically rising. TQuel incorporates the function RATE which compares two values of the same instance over a certain time period. For example, it could determine what values have doubled over the last month. HSQL has new built-in functions as FIRST and LAST, that return the earliest and latest time point at which a certain condition is valid.

2.4.5 Vacuuming

As mentioned earlier, transaction databases can grow very large, because data is only logically deleted. However, this can become a major problem as each system has limited storage capacity. In addition, huge amounts of data that is only rarely accessed could severely compromise performance. Another problem is that in many countries law forbids certain information to be retained longer than a specific time interval. Therefore temporal databases have support for *physical deletion* of data. This is called *vacuuming*.

2.4.6 Implementation issues

Temporal database implementation involves specific implementation issues. Two important aspects are:

- data dictionary,
- query optimization.
- Efficient storage

The data dictionary differs from the traditional database because first, in transaction time databases different versions of the meta data must be supported (i.e. schema versioning). Second, calendars must be supported, which define a mapping between time units with different time granularities. In addition reasoning with different modalities must be supported.

Query optimization is more involved in TDBs. Two reasons for this are the increased size of the relations and the increased complexity of the logic. The increased size of the relations is due to the fact that temporal relations have time elements associated with them. In addition, transaction databases also retain previous states of these relations, which makes the size grow monotonically. The increased complexity of the logic makes query optimization harder. In conventional systems the emphasis is mainly on equality predicates.

Certain characteristics of temporal relations may also help to improve efficient query optimization. An example is the fact, that time advances in one direction only. This implies an ordering on temporal relations which can be exploited when evaluating for example extremes. In addition, recent information is accessed much more frequently than historical information. It is also probable that the access patterns differ as well. *Temporal partitioning* can then be adopted to separate the historical data *physically* from the current data while preserving a logical overall picture. Another important optimization technique are temporal indexes, where these special temporal characteristics can be exploited to gain performance.

Optimizing the physical storage model for temporal databases is an important issue as the amount of data in a temporal databases increases at high speed (especially when transaction time is supported and data is not often deleted physically). Unfortunately we could not find enough relevant material on this issue in the literature.

2.5 Adjoining database areas

The goal of this paragraph is to place temporal databases in a broader context. To achieve this, some related topics are described in a general matter without going into detail because that is out of the scope of this document. First some combinations of temporal databases with some other database aspect are described. For these types of databases the same applies as for temporal database; they are still in their infancy and no commercial products are available.

2.5.1 Spatio-temporal databases

A spatio-temporal database supports, beside time, the extra dimension of *space.* In fact, a better way of defining is a spatial databases extended with time, as this is this case in most real-life situations. Two main areas that can use this type of database are Geographic Information Systems (GIS) and Multi Media systems.

Geography has three fundamental properties (Figure 2.6)
- Location (place)
- Attribute
- Time

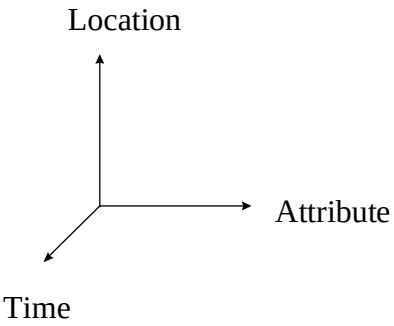


Figure 2.6 Fundametal properties of geography

Location describes where an object or phenomena is situated (using spatial coordinates like e.g. x,y,z). *Attributes* describe properties of the object (e.g. it is *hot* and *dangerous*). *Time* denotes the change of an object or phenomena over time. As with ordinary databases, a lot of GIS databases do not contain *time* as a fundamental property (equal to Location and Attribute) but more as a special type of attribute. This way of time modelling is insufficient to fully support basic GIS areas as *Dynamic modelling (simulations, predictive modelling)*, *Continually updated GIS data structures over time* and *Representing rapidly changing objects or real time analysis* because of the importance of time in these areas.

As with temporal databases, the concepts of *valid time* (in this context also refered to as *physical time*) and *transaction time* can be used to model time (or to speak with [Wilson96], to model "*reality* versus *representation*"). Physical time is either the moment an object was made or the property of an object (e.g. this hurricane originated at July 1, 1997 or this fossil has been determined to be two million years old). Transaction time is defined as in an ordinary temporal database.

For Multi Media databases the same concepts can be used. In fact, multi media databases are very well suited in GISs. As an example take a multi media database containing football videos. The ball is an object with a location and attributes over time. To determine whether there has been a score or not, one has to determine whether the ball was at the same location as the goal for a certain period of time.

2.5.2 Temporal deductive databases

In an ordinary temporal database, all occurences of a relation (the so called *extension*) over time have to be explicitly stored. The result of this is a huge database, especially when transaction time is supported. To limit the required storage capacity the extension can also be represented implicitly by means of deductive rules. For these rules, Horn-clauses (without function symbols) can be used . Beside limiting storage requirements, implict representation also enables to represent infinite extensions. A temporal database using this way of implicit representing extensions is called a Temporal Deductive Database.

2.5.3 Temporal real-time databases

The usual definition of a real-time database is "a database with deadlines for transactions". Another way of defining is "a time-constained database" [OS95]. These time constrains can apply to all aspects of the database e.g. reponse to queries; processing insert, update, delete transactions; maintaining integrity etc. From the definition, one can see the importance of *time* in this type of database. Current real-time systems use their own (limited) definition of time, but using the well defined time concepts from the temporal database field would be an enrichment for these systems (e.g. enabling reasoning about the future).

Valid time is used for data items that have immediate counterparts (external objects) in the real (physical) world. External events resulting in value changes for these external objects are closely monitored and a transaction, including the valid time of occurrence of this event, is written to the database (e.g. a sensor detecting that the temperature in a room is above some critical level). Transaction time is used for transactions that, with the help of some device, set parameters of the real-time system (e.g. activating a motor to open the window in order to decrease the temperature). Furthermore, transaction time is used when new values for data items are derived from values of existing data items.

3 A temporal query language

In the previous chapter, the principals of temporal databases have been described. The subject of chapter 3 is a query language for temporal databases. As mentioned in chapter 2, most temporal database proposals and prototypes are extentions of the relational model. For this reason, we limit ourselves in this chapter to temporal extensions of SQL-92.

Several movements have led to the current existing most promising temporal query languages. On one hand there was the *TSQL2 Language Specification* as published in September 1994 [TSQL94]. TSQL2 is a bitemporal language supporting *valid time* and *transaction time* (in addition to the already existing (poor) support for user-defined time). On the other hand, the Knowledge Based Systems Group at ETH Zürich, constructed an interval timestamped temporal deductive database system called *ChronoLog* [Böhlen95a]. The ChronoLog system supports *ChronoLog*, a temporal extension of first order predicate logic, and *ChronoSQL*, a temporal extension of SQL. In contrast with TSQL2, ChronoLog is an actual existing prototype system.
The ChronoLog and TSQL2 people joined forces in the *Advanced TSQL2* (ATSQL2) project, combing theory and practice. With knowledge gained in this project, expert contributions where made for the ISO/ANSI *SQL/Temporal* module of the SQL3 standard [ANSI96a], [ANSI96b]. The SQL/Temporal module is on hold for the moment.
After the ATSQL2 project, Böhlen and Jensen continued their work on temporal databases with the *Tiger* system at the university of Aalborg in Denmark. The Tiger system is a temporal front-end on top of the Oracle RDBMS. The Tiger system is accessible from the WWW and has a query language called ATSQL [http://www.cs.auc.dk/~tigeradm/]. Another person from the ATSQL2 project, A. Steiner, started the development of a commercial TDB called *TimeDB*. TimeDB implements selected aspects of SQL/Temporal. Like Tiger, TimeDB is a temporal front-end on top of an existing RDBMS. Version 1.0 was still a general available prototype, but version 2.0 (due in 1999) will be a Java based commercial version that can be connected to all RDBMSs supporting JDBC [http://www.timeconsult.com/TimeConsult.html].

The discussion of a temporal query language in this chapter is based on material from the above mentioned movements. Because of availability of the Tiger system, the examples are based on ATSQL. ATSQL is an implementation of the SQL/Temporal proposals. Whereas SQL/Temporal is on hold for the moment, the development of ATSQL continues.

3.1 Upward compatibility with SQL-92

For new technology to be accepted, compatibility with predecessor technology is important. ATSQL and the SQL/Temporal proposals fulfill this obligation because they are fully upward compatible with ordinary SQL i.e. all existing application code will correctly work with the temporal database without needing modification. To be honest, we have to nuance the compatibility issue. A lot of current applications already maintain time-varying data using non-temporal databases. Time has explicitly been modelled in the schema and the application contains code to reason on this data. When upgrading the non-temporal database to a temporal database, in theory, it is not necessary to change schema's or application code. But this will not change the situation as the applications do not use the capabilities of the temporal databases. To exploit these capabilities, the schema's and applications have to be changed. But, as this change can

be done in a gradual way, compatibility stays an important issue. Non temporal application code can coexist with temporal application code.

3.2 Time ontology

The discussion whether time is continuous, dense or discrete is carefully avoided by TSQL2. All three are supported by assuming that a time instant, that is much smaller than a chronon, always has to be estimated by a chronon because a chronon is the smallest addressable unit of time. At runtime, timestamps are associated with a user-specified granularity (e.g. second, hour, day etc). A temporal query is always stated in terms of some granularity viz. the question whether a period x is before, during of after a period y can only be asked in terms of a same granularity.

3.3 Base line clock

A semantics to time is given by using a so called *base line clock*

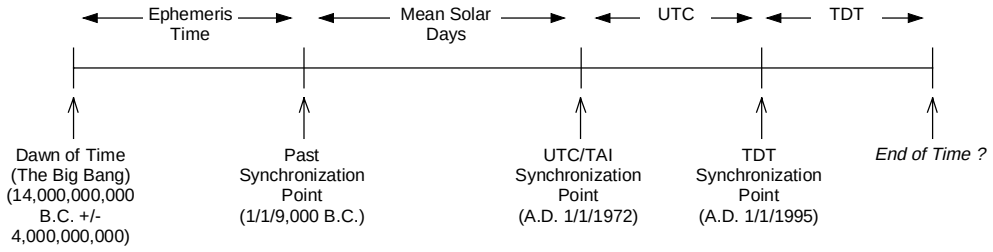


Figure 3.1 Base Line Clock

This base line clock is bounded on two sides; it starts 18 billion years ago and it extents 18 billion years into the future. It is partitioned in a set of contigeous periods based on historical events. The introduction of atomic clocks made time measuring very precise and Universal Time Coordinated (UTC) was introduced. Nowadays Terrestrial Dynamic Time (TDT) is prefered over UTC because it is even more precise also taking leap seconds into account.

3.4 Data types

TSQL2 adds the datatype *Period*[13] to the already existing SQL-92 datatypes *Date, Time, DateTime* and *Interval*[14]. A period is a set of two time instants with the constraint that the instant that starts the period equals or precedes the instant that terminates the period. Remind that comparison of time data is always in terms of some granularity. Finally, there is a datatype called *surrogate*. A surrogate is a unique identifier, not visible to the user, used to determine identity of objects. This is useful when e.g. the primary key is time-varying.

3.5 Table types

ATSQL and the SQL/Temporal proposals distinguish four types of tables depending on the time elements associated with its tuples:

- snapshot tables: having no temporal support except for user-defined time.
- valid-time tables: having a valid-time period associated with each tuple.
- transaction-time tables: having a transaction-time period associated with each tuple.
- bitemporal tables: having both a valid time period and a transaction time period associated with each tuple.

---

13 In the TDB glossary this is called a Time Interval.

14 In the TDB glossary this is called a Time Span.

Different table types are allowed in one database schema. Note that the time elements are not visible as simple attributes of a tuple although it is possible to ask for the associated time values of a tuple by means of special functions (VTime() and TTime()). The time periods are closed intervals. Besides “ordinary” values for the time elements, there are also some special values. For valid-time and user-defined time the values *beginning* and *forever* may be used to indicate the least and the greatest values respectively. As an extra, valid-time and user-time can be temporally indeterminate i.e. and event has occured but it is not clear (yet) when. How to handle this uncertainty can be determined on a per-query-basis or on a global basis. Finally, the concepts *Current* and *Now* are introduced. *Current* is immediately substituted by the current date and time. The actual value of *Now* is determined at runtime when a query is executed, so it is dynamically changing.

3.6 Levels of temporal functionality

For supporting valid time in SQL/Temporal, a four level approach is proposed in [ANSI96a], [ANSI96b] and [Tiger98]. Each level adds more functionality to the former level. A series of figures is added for illustration.

Level 1: Upward Compatibility (UC)

This level provides functionalities to be upward compatible with SQL3. In fact, this is not yet a temporal level. Figure 3.2 shows the current state of a table (a solid rectangle) in the upper right corner. To the left of this, dashed rectangles represent the history of the table that is not in the database anymore (because there is no temporal support yet). A query q, queries the current state of the table resulting in one output set.

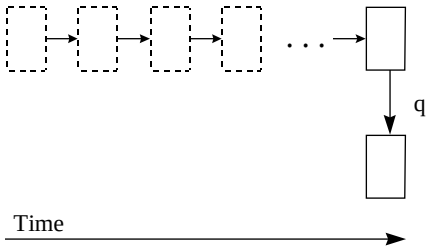


Figure 3.2 Upward Compatibility (UC)

Level 2: Temporal Upward Compatibility (TUC)

This level adds to the former level the functionality to define valid time and transaction time for tables. It does not yet extend the query language to use these time properties viz. only regular SQL3 query statements can be used. The following extentions of SQL3 DDL statements are added:

```
CREATE TABLE <table_name> ( .. ) AS VALID;
CREATE TABLE <table_name> ( .. ) AS TRANSACTION;
CREATE TABLE <table_name> ( .. ) AS VALID AND TRANSACTION;
ALTER TABLE <table_name> ADD VALID;
ALTER TABLE <table_name> ADD TRANSACTION;
```

When adding valid time to existing tables, the valid time period for each tuple current in the table is set to *[current - now)*. The same applies to transaction time. After a table has been made temporal, querying that table with a normal SQL3 statement, gives the same (snapshot) result as querying a non-temporal version of that table. This is illustrated in Figure 3.3. As there is temporal support, the history is present in the database (solid rectangles), but only current state tables can be queried.

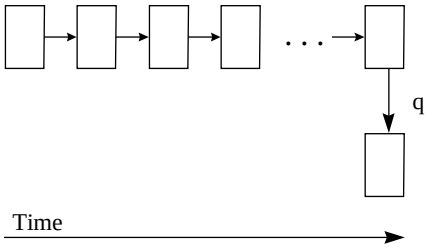


Figure 3.3 Temporal Upward Compatibility (TUC)

Level 3: Sequentiality (SEQ)

At this level, the query language is extended to give so called *sequenced* temporal functionality to queries, views, constraints, assertions and modifications on tables with valid time and transaction time support. *Sequenced* means applying the query to all available states of a table and not only to the current state. Figure 3.4 shows a query q' which is the equivalent of applying query q to all states of the table resulting in an output set for each state. Remark that for each output set only the accompaning table state can be used.

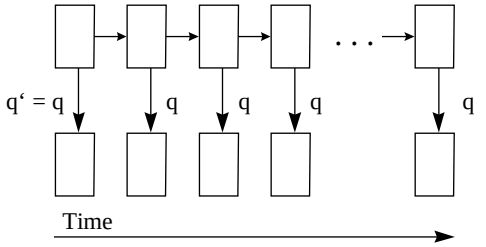


Figure 3.4 Sequentiality (SEQ)

To enable sequenced functionality the query language is extended with the reserved word SEQUENCED. This word has to be prefixed to a query e.g.:

```
SEQUENCED VALID SELECT * FROM Customer;
SEQUENCED TRANSACTION SELECT * FROM Customer;
SEQUENCED VALID AND SEQUENCED TRANSACTION SELECT * FROM Customer;
SET VALID PERIOD '1995 - 1998' INSERT INTO Customer VALUES (..);
```

The result set of a sequenced query contains the explicitly asked (non-temporal) attributes together with the valid time and/or transaction time period (depending on whether VALID, TRANSACTION OR VALID/TRANSACTION was specified). When a bitemporal table is queried with SEQUENCED VALID, the result is so called *Temporal Upward Compatible (TUC)* in the transaction time dimension and *Sequenced* in the valid time dimension. This means that tuples with transaction time periods that ended before the current date are not visible. In the same way, a SEQUENCED TRANSACTION query is TUC in the valid time dimension and *Sequenced* in the transaction time dimension.

Level 4: Nonsequentiality (NONSEQ)

At this level, the query language is extended to give so called non*sequenced* temporal functionality to queries, views, constraints, assertions and modifications on tables with valid time and transaction time support. *Nonsequence*d means applying the query to all available states of a table in the database. In contrast to a sequenced query, each state of the resulting relation may use information from all other states in the database (Figure 3.5).

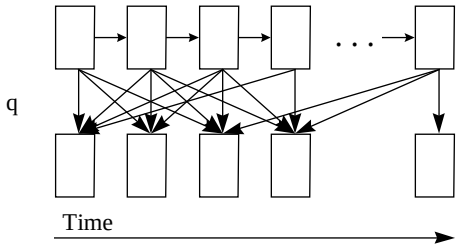


Figure 3.5 Nonsequentiality (NONSEQ)

To enable nonsequenced functionality the query language is extended with the reserved word NONSEQUENCED. This word has to be prefixed to a query e.g.:

```
NONSEQUENCED VALID SELECT * FROM Customer;
NONSEQUENCED TRANSACTION SELECT * FROM Customer;
NONSEQUENCED VALID AND SEQUENCED TRANSACTION SELECT * FROM Customer;
```

Queries with nonsequenced semtantics treat the time elements as any other user defined attribute. The result set of a nonsequenced query only contains the explicitly asked (non-temporal) attributes. The valid and transaction time period are not in the result set. To get these elements, the special functions *VTime(<table>)* and *TTime(<table>)* have to be used. In fact, the data of all time periods is available to the user and all this data can be used, but the user himself has to specifiy how to treat the available time in this data (by means of the VTime() and TTime() functions). The DBMS does not interpret the time elements.

3.7 ATSQL examples

In this paragraph, some temporal queries are described to give an idea of how temporal queries look like. Remark that it is not the intention of the paragraph to give a complete overview of all temporal query statements. The examples are expressed in the ATSQL syntax as supported by the *Tiger* prototype temporal database.

A first example shows how the upward compatability works. Suppose we already have a (non temporal) table *Customer* in our database. The following statements adds temporal support to this table without affecting the snapshot behaviour of the table.

```
Alter table Customer add valid;
Alter table Customer add transaction;
```

All (existing) data in the table is valid from the current date until forever and the same applies for transaction time. A select statement on this table will yield the same result as a select statement on the non temporal table. If required, temporal support can be removed with the following statements.

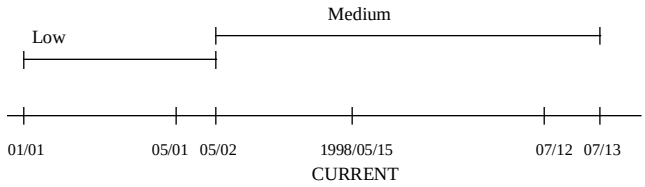
```
Alter table Customer drop valid;
Alter table Customer drop transaction;
```

The following example creates a new table with both valid-time and transaction-time support and inserts two tuple in this table. Assume the current date is 1998/05/15.

```
Create table Customer (   Name varchar(30), Priceplan varchar(20),
                          Turnover_cat varchar(10))
as valid and transaction;

Set Valid period '1998/01/01-1998/05/01'
insert into Customer values ('Temp Db Inc.', 'Budgetplan', 'Low');
Set Valid period '1998/05/02-1998/07/12'
insert into Customer values ('Temp Db Inc.', 'Budgetplan', 'Medium');
```

The transaction time of both tuples, generated by the system, is [1998/05/15-now). Graphically, the valid time of the tuples can be represented as follows:



For retrieving the stored information, several alternatives are available.

First, a Temporal Upward Compatible (TUC) select-statement:

```
Select * from customer;

NAME         PRICEPLAN        TURNOVER_CAT
-----------------------------------------
Temp Db Inc. BudgetPlan       Medium
```

The result is that only the second tuple is shown because that is the only tuple that is valid at this moment (i.e. 1998/05/15) and in the database according to the transaction time.

Second, a Sequenced valid time example is given. In this case, the transaction time is treated TUC in other words, only tuples where the current date is in the transaction time periode are eligible for the query. The statement for this is:

```
Sequenced Valid Select * from Customer;

VT                    NAME               PRICEPLAN      TURNOVER_CAT
-------------------------------------------------------------------
01-JAN-98-01-MAY-98  Temp Db Inc.       Budgetplan     Low
02-MAY-98-12-JUL-98  Temp Db Inc.       Budgetplan     Medium
```

In this case, the two inserted tuples are shown because interpretation of the transaction time tells that these facts would be in the database from 1998/05/15 until now, and our current date is in this period.

Third, a sequenced transaction time example is given. In this case, the valid time is treated as TUC in other words, only tuples where the current date is in the valid time periode are eligible for the query. The statement for this is:

```
Sequenced Transaction Select * from customer;

TT             NAME                 PRICEPLAN        TURNOVER_CAT
-----------------------------------------------------------------
15-MAY-98-NOW  Temp Db Inc.         Budgetplan       Medium
```

The result is that only the second tuple is shown because interpretation of the valid time tells that only this tuple would be valid and thus in the database at this moment.

Fourth, a combined sequenced transaction and valid time example is given. This statement will allways show all tuples in the databases. The statement for this is:

```
Sequenced Valid and sequenced transaction select * from customer;

TT             VT                    NAME         PRICEPLAN TURNOVER_CAT
-----------------------------------------------------------------------
15-MAY-98-NOW  01-JAN-98-01-MAY-98  Temp Db Inc.  Budgetplan     Low
15-MAY-98-NOW  02-MAY-98-12-JUL-98  Temp Db Inc.  Budgetplan     Medium
```

For this statement, the current date is not relevant because all tuples in the database are shown. One can also see here, that Tiger uses time spans for both the valid time and the transaction time. When a tuple is inserted, the transaction time is *[current_date - now)*. When a tuples is (logically) deleted from the database, the end time of the transaction time changes from *now* to the transaction time of the time of deletion.

In the case of the simple example select statements, nonsequenced statements would return the same tuples but without the valid time and transaction time values in the result set e.g.:

```
NonSequenced Valid and Nonsequenced transaction select * from customer;

NAME                          PRICEPLAN            TURNOVER_CAT
-------------------------------------------------------------------
Temp Db Inc.                  Budgetplan           Low
Temp Db Inc.                  Budgetplan           Medium
```

If the users wants the valid time and transactin time values, he has to explicitely retrieve them using the VTime() and TTime() functions e.g.

```
NonSequenced Valid and Nonsequenced transaction
select ttime(customer), vtime(customer), name, pricplan, turnover_cat
from customer;

TT             VT                    NAME        PRICEPLAN TURNOVER_CAT
-------------------------------------------------------------------------
15-MAY-98-NOW  01-JAN-98-01-MAY-98  Temp Db Inc.  Budgetplan  Low
15-MAY-98-NOW  02-MAY-98-12-JUL-98  Temp Db Inc.  Budgetplan  Medium
```

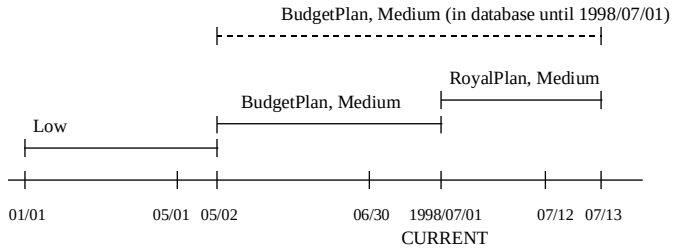
Suppose, at 1998/07/01, customer *Temp Db Inc.* gets a “RoyalPlan” priceplan. We use the following update statement to store this information in the database:

```
update Customer
set  priceplan  = ‘RoyalPlan’
where name =’Temp Db Inc.’;

Sequenced Valid and sequenced transaction select * from customer;

TT                    VT                   NAME    PRICEPLAN  URNOVER_CAT
-------------------------------------------------------------------------
15-MAY-98-NOW         01-JAN-98-01-MAY-98 Temp Db Inc. Budgetplan Low
01-JUL-98-NOW         01-JUL-98-12-JUL-98 Temp Db Inc. Royal Plan Medium
15-MAY-98-30-JUN-98   02-MAY-98-12-JUL-98 Temp Db Inc. Budgetplan Medium
01-JUL-98-NOW         02-MAY-98-30-JUN-98 Temp Db Inc. Budgetplan Medium
```

Graphically, the valid time of the tuples can be represented as follows:



Here we see some typical temporal behaviour. As we do a snapshot update, only the tuple valid at the current time (1998/07/01), that is the one with turnover_category “Medium”, is involved. As this table has transaction support, the existing value is not overwritten. Instead the transaction time period is closed at 1998/06/30 (this tuple is depicted in the figure with a dotted line). In fact, this tuple is logically deleted. Two new tuples are added to depict the new situation. As we did not specify a valid time period in our update statement, the valid time period for the “RoyalPlan”-tuple is set from the current date (1998/07/01) until the original valid enddate (1998/07/01). In the period 1998/05/01-1998/30/06, the “BudgetPlan” value was valid.

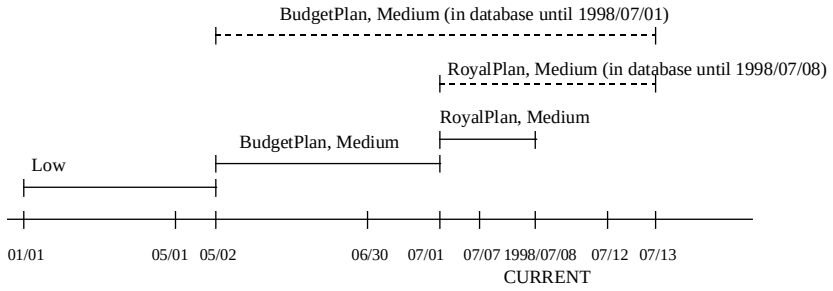
Up to now we illustrated temporal SELECT, INSERT and UPDATE, so now it is time for a DELETE. Suppose, at 1998/07/08, our “Time Db Inc.” customer goes bankrupt and we

want to delete his data from our database. As with the update, we use a snapshot statement for this action because we only want to change the current situation.

```
delete from customer where name='Time Db Inc.';
Sequenced Valid and sequenced transaction select * from customer;

TT                  VT                    NAME   PRICEPLAN  TURNOVER_CAT
-------------------------------------------------------------------------
15-MAY-98-NOW       01-JAN-98-01-MAY-98  Temp Db Inc. Budgetplan Low
01-JUL-98-07-JUL-98 01-JUL-98-12-JUL-98  Temp Db Inc. Royal Plan Medium
15-MAY-98-30-JUN-98 02-MAY-98-12-JUL-98  Temp Db Inc. Budgetplan Medium
01-JUL-98-NOW       02-MAY-98-30-JUN-98  Temp Db Inc. Budgetplan Medium
08-JUL-98-NOW       01-JUL-98-07-JUL-98  Temp Db Inc. Royal Plan Medium
```

Graphically, the valid time of the tuples can be represented as follows:



As we have transaction support, the current “RoyalPlan”-tuple is not physically deleted from the database but only logically (dotted line). The transaction time period is closed at 1998/07/07. Furthermore, the new situation that our customer had a RoyalPlan from 1998/07/01 until 1998/07/07, with a transaction time period from [1998/07/08-now), is decuded and stored by the database.

The last temporal concept illustrated in this paragraph is coalescing. Coalescing involves joining of adjacent or overlapping time periods. Coalescing can both be done for valid time periods and for transaction time periods. In this example we limit ourselves to coalescing in valid time. Suppose we want to know in what period our customer had a “BudgetPlan”-priceplan independent of the turnover category. As this is a question associated with history data, we use atemporal query:

```
sequenced valid
select name, priceplan from customer where priceplan='BudgetPlan';

VT                  NAME                PRICEPLAN
-----------------------------------------------
01-JAN-98-01-MAY-98  Temp Db Inc.        Budgetplan
02-MAY-98-30-JUN-98  Temp Db Inc.        Budgetplan
```

The result contains as aspected two tuples. But infact it concerns two adjacent periods where the priceplan stayed the same but the turnover category changed fron “Low” to “Medium”. As we were interested in the priceplan independent of the turnover category, we would like to see a one tuple result with a valid period [1998/01/01-1998/06/30]. To achieve this, we perform a coalescing operation in the valid time dimension.

```
(sequenced valid
select name, priceplan from customer where priceplan='BudgetPlan')(vt);

VT                  NAME                            PRICEPLAN
-------------------------------------------------------------
01-JAN-98-30-JUN-98  Temp Db Inc.                    Budgetplan
```

4 Tools and products

After twenty years of research into temporal databases, temporal support in commercial databases is still very limited. The fact that ATSQL is a superset of SQL92 and that a prototype and a commercial ATSQL implementation have been realised as a front-end to existing major commercial DBMSs, has not convinced the DBMS vendors of incorporating temporal support into their databases. Users of the DBMSs of major vendors, do not ask for temporal support yet probably because the added value of temporal support is not clear to them. When users ask for temporal support, DBMS vendors (e.g. Oracle and Informix) say they already have it in the form of so called *Time Series*. This chapter will first give an overview of the prototypes that have been developed. After that the concept of Time Series is explained.

4.1 Prototypes

In [Böhlen95] an overview is given of temporal prototypes that have been developed until 1995. Although this list may be outdated, we think that it still gives a good indication about the implementation efforts in this area. The extent to which the implementations support standard and temporal database functionality varies much. Several implementations even lack traditional database functionality like persistence, transactions and concurrency control. In our opinion the most interesting prototypes are the front-end implementations for commercial databases (e.g. Tiger and TimeDB[15]). In Table 4.1 an overview is given of the temporal prototypes as described in [Böhlen95].

Name[16]	1	2	3	4	5	6	7	8	9
Tiger/TimeDB	R	TS	UVT	T	CG[17]	PTC	SV	SUPJN	QUI
ChronoLog	R	TS	UVT	T	-	PTC	SV	SUPJN	QUC
HDBMS	R	TS	UVT	T	G	PTC	S	SUPJN	QU
TempIS	R	TS	UVT	T	-	PTC	S	SUPJ	QU
VT-SQL	R	TS	UV	T	-	PTC	-	SUPJN	QU
TDBMS	R	TS	UV	TA	P	PT	S	SPJN	QU
T-squared DBMS	R	S	UVT	T	-	P	V	SUPJN	QU
T-REQUIEM	R	TSE	UVT	T	-	P	SV	SUPJ	Q
TIMEIT	R	S	V	T	-	-	-	J	Q
TIMEMULTICAL	R	TS	U	-	CGI	-	S	S	Q
Calanda	O	T	V	-	CGI	PTC	S	S	Q
ARCADIA	O	TS	UV	-	GI	P	-	-	QM
TempCASE	-	TSE	UV	-	G	P	OSV	SUPJN	QU

Table 4.1 Salient features of TDB prototypes (taken from [Böhlen95])

Legend[18]:
1. relational (R), object-oriented (O),
2. points (T), spans (S), set of spans (E),
3. user-defined time (U), valid-time (V), transaction time (T),
4. tuple time stamps (T), attribute time stamps (A),

[15] In 1999 a commercial, platform and database independent version of TimeDB will come out.

[16] The prototypes are ordered in decreasing order of amount of functionality. The amount of functionality is determined by weighing column 6, 3 and 8 in decreasing order of importance.

[17] Is planned for implementation.

[18] The number refers to the column in the table.

5. multiple calendars (C), different granularities (G), indeterminacy (I), interpolation (P),
6. persistence (P), transactions (T), concurrency control (C),
7. object versioning (O), schema modifications (S), views/rules (V),
8. temporal selection (S), temporal union (U), temporal projection (P), temporal join (J), temporal negation (N),
9. temporal queries (Q), temporal updates (U), temporal integrity constraints (C), temporal methods (M)

Table 4.1 shows that most prototypes are based on the relational model. In addition almost all prototypes support both time span and time points representation. The support for temporal elements (i.e. multiple valued spans) is negligible. Column three shows that almost all prototypes support user-defined and valid- time support. Transaction time support is much less frequent; about half of the implementations. When we look at the relational based models, we see that the time stamping is modeled at tuple level. Only TDBMS supports both tuple and attribute time stamping.

Support for reasoning at different time granularities and associated modalities is only rarely supported. Of the traditional database characteristics persistence, transactions and concurrency control is persistence widely supported. Transactions and concurrency control is supported by only some of the prototypes. Column 7 is not clarified by [Böhlen95]. Column 8 indicates that most implementations support temporal Select, Project Join (SPJ) functionality. In addition most implementations support temporal Union and Negation. Temporal querying is supported by almost all prototypes, whereas temporal updates are only supported by only some of the implementations.

Table 4.1 is ordered in descending order on completeness of temporal support. Completeness is rated on columns 6, 3 and 8 in descending order of importance. In addition the prototypes are categorized on underlying data model (i.e. column 1).

The prototypes that have the most complete functionality are Tiger/TimeDB, Chronolog, HDBMS and TempIS. Notably, Tiger/TimeDB, Chronolog and TempIS are front-ends to commercial databases (i.e. Oracle or Ingres). HDBMS is an independent implementation which covers database functionality by itself.

The front-end implementations prove that it is possible to extend commercial products with temporal support. However, as mentioned before, market leaders like Oracle and Informix are not interested (yet) in extending their products with temporal functionality.

4.2 Time Series

We have already mentioned that commercial products do not have temporal support as defined in the previous paragraphs. However, they sometimes do provide temporal extensions to their standard database engine targeted at specific application area's. This specific temporal support is covered by the term *time series*. This paragraph will explain the term *time series* and its relation to temporal databases.

The short definition of a time series is; *a collection of observations made sequentially over time* (e.g. the hourly price of a share). Time series are often used in areas like banking and meteorology. Several thousands of time series to forecast economic parameters or the weather are very common. Special support is required for storing and manipulating these amounts of time series.

4.2.1 Modelling of time series

For a data model to support time series, the following structural elements are necessary [Schmidt95]:

- **Events**: the actual registration of the value(s) of interest. These are the building blocks of time series. The values can be single-valued or multi-valued. Furthermore, there is a distinction between so called *base values* i.e. measured facts and *derived values* i.e. values computed from base values.

- **Time series**: a sequence of chronologically ordered events preceded by a header containing common data for the entire time serie. The header may contain both time-variant data (e.g. average price of a share) and time-invariant data (e.g. the name of the time serie). The events of a time series may be of the same type or vary over time.
- **Groups of time series**: a grouping of time series according to some criteria (e.g. country, branch) in order to facilitate the manipulation of large sets of time series. Each group has a header with common data for the entire group, and a set of members. These members can be both time series as other groups. Group members can belong to different groups at the same time.
- **Calendars:** a set of valid times at which values are measured for a time series. There are three types of calenders viz [Etzion98]:
  - Calendars modelling physical space (e.g. Days, Hours, Minutes, Seconds),
  - Calendars defined in accordance with a particular calendar system. Example of calendar systems are Gregorian, Jewish and Islamic. Calendars *Years, Months, Weeks* are examples of Gregorian calenders.
  - Calendars defined in accordance with particular applications (e.g. a calendar with holidays). This type of calendar is user-defined whereas the two previous types are system defined.

In Figure 4.1, an example of a time series for recording share prices is given. Four times a (business) day, the actual price of a share is measured and beside these values the lowest and highest value of each day are recorded separately. The header contains the name of the time series, the calendar used and a (at forehand unknown) number of values e.g. for statistical purposes.

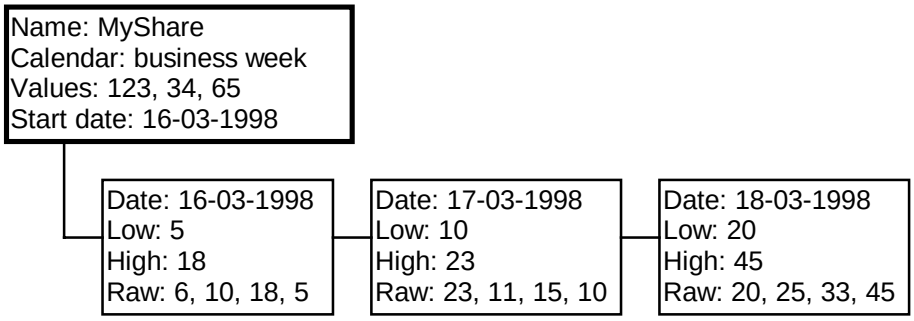


Figure 4.1 Example of a share price time series

Time series are mostly accessed along the time axis, event by event. But when so called *cross-sectional* analysis is performed, several time series are accessed simultaneous comparing the different events from each time series at the same points in time. The data model should support efficient storage for both types of access.

4.2.2 Operations on time series

For manipulation of time series, the usual CRUD[19] operations on simple and complex type elements should be supported. Another important operation is the derivation of new time series from existing ones e.g. by computing the difference of two existing time series or transforming the periodicity of a time series (e.g. daily periodicity to monthly periodicity). As time series are often subject to statistical analysis, requiring matrix algebra, operations on (multidimensional) arrays should be supported. Periodicity transformation is not always trivial. Take for example the transformation from daily to monthly for a share price. For the *high selling price* it is the maximum of all daily selling prices but for the *closing price* it is the closing price of the last day of the month. To facilitate working with *groups*, additional operations are required for creating and manipulating groups of time series.

[19] Create/Read/Update/Delete

4.2.3 Time series and temporal databases

After the above description of the requirements for modelling and manipulating time series, the question arises whether this can be implemented in a temporal database or not. The other way around, the question is whether a Time Series Management System (TSMS) is the same as a Temporal DataBase Management System (TDBMS).
In a temporal context, one can distinguish three types of objects:

- Time-invariant objects, which values do not change over time (e.g. date of birht of a person)
- Time-varying objects, which values change over time with arbitrary frequency (e.g. the priceplan of a customer)
- Time-series objects, which values change over time with a frequency imposed by the associated calendar (e.g. a share price time series).

Both TSMSs and TDBMSs support time-invariant objects but TSMS are focussed on time-series object support whereas TDBMSs are focused on time-varying object support. A TSMS is aimed at efficient storage of (groups of) huge arrarys of (multi-valued) elements and provides fast implementations of dedicated operations to manipulate these elements. For time-series, only the notion of valid time is relevant. In a TDBMS, on the other hand, both valid time and transaction time should be supported and upward compatibility with existing applications is desirable.

We think that it is possible to use a TDBMS as TSMS but, as a TDBMS is a general solution, it will not be as efficient and fast as a dedicated TSMS. Furthermore, some additional application programming is necessary to support the required operations. For modeling it is important whether a relational TDBMS is used or a Object Oriented one. The latter is more easy as it can handle the complex data types used with time series. In a relational database, complex types can be modelled using several related relations. As most temporal databases are extensions of the relational model, we will give some relational approaches, and their drawbacks, for modelling time series [Schmidt95]:

- A separate relation for each time series, related to another relation for the header information. For a group the same strategy can be used with a relation with the group header information related to a relation containing pointers to the time series header and group header relations belonging to this group. As time series bases usually consist of thousands of time series, the result is an enormous amount of relations that are difficult to manage.

TS_11

Timestamp	Attr-1	Attr-2	…
1	123	“xyz”	…
2	456	“abc”	…

TS_12

Timestamp	Attr-A	Attr-B	…
1	78.98	89	…
2	45.9996	8976	…

- A separate relation for all time series of the same type with an attribute indicating the id of each time series (i.e. each time serie of a certain type is a tuple in a relation). For groups, the same kind of strategy as above can be used. The result is less relations, but conflicts in reading and writing performance. As time series are mostly sequentially read, sorting the relation by time-series_id, date/time is desirable for reading. Normally, time series are rarely update; only new events are appended at the end of a time series. This results in frequent rewriting of indexes with the above sorting strategy.

TS_type_X

Timestamp	ts_id	Attr-1	Attr-2	…
1	“TS_11”	123	“xyz”	…
2	“TS_11”	456	“abc”	…
1	“TS_14”	909	“hoi”	…
2	“TS_14”	888	“mno”	…

- Another approach is to use one relation for all time series. This results in an enormous waste of storage space as this relation must contain all attribute of all time series and most of the fields have a null value

TS_Events

Timestamp	ts_id	Attr-1	Attr-2	Attr-A	Attr-B	…
1	“TS_11”	123	“xyz”	null	null	…
2	“TS_11”	456	“abc”	null	null	…
1	“TS_14”	909	“hoi”	null	null	…
2	“TS_14”	888	“mno”	null	null	…
1	“TS_12”	null	null	78.98	89	…
2	“TS_12”	null	null	45.9996	8976	…

Remark that the number of relations is further increased in the case of multi-valued attributes. A drawback from a query point of view is the fact that relational databases are set oriented whereas time series are sequential oriented (e.g. give me the 10^{th} and 40^{th} element of a time series). This may lead to complex queries with long response times.

Using a temporal relational database to implement a time series application instead of a conventional snapshot relational database has the advantage of the support for valid time. But a temporal database does not fully support times series. The absence of complex data types and specific time series operations, requires a lot of functionality to be implemented by the user.

The choice between a TDBMS and a TSMS has to do with the difference between a general or a dedicated solution for a particular problem. A TSMS is tailored for time series management but does not support other types of data manipulation whereas a TDBMS is designed for more general time manipulation.

4.2.4 Time series products

As explained in the former paragraph, relational temporal databases have their limitations concerning the support of time series. Current temporal databases do not explicit support time series management. As already indicated above, the first alternatives are provided by research prototype and commercial object oriented or object relational database management systems. An example of the former is the CALANDA TSMS developed at the Union Bank of Switzerland. Commercial time series support is offered by ORDBMSs from, among others, Oracle and Informix. The time series data type, including accompanying operations, is provided by special plug-in modules (called *cartridges* for Oracle and *data blades* for Informix). These modules extend the database with array based storage of (complex type) elements. Furthermore operations are provided for creating time series and efficient operations for manipulation of these time series. The concept of grouping time series is not yet supported. A more detailed description of the capabilities of these plug-ins is out of the scope of this document.

5 Analysis and recommendations

Analysis

After some twenty years of temporal database research, consensus on concepts and aspects has been reached within the temporal database community. The concepts of Valid Time (*when is a fact true in reality*) and Transaction Time (*when is a fact stored in the database*) are considered the cornerstones for temporal support. As compatibility with existing database systems has high priority, most researchers have tried to extend the relational or the object oriented model. Extensions of the relational model where the most succesful probably because most current databases are relational databases. The ideas and concepts of the ATSQL project are the most likely candidates to be incorporated in the Temporal module of the new SQL3 standard. Comparing functional equivalent standard SQL queries and temporal SQL queries indeed shows that temporal SQL queries are much easier and powerful than standard SQL queries. Furthermore, the datamodel can be simplified by removing 1:N-constructs for storing history data.

Up to now, major database vendors like Oracle, Informix, IBM and Microsoft do not show real interest for temporal support in their products althought prototype and commercial front-ends have shown the possiblities of it. When asked for it, the vendors refer to their support of time series. But time series are specialized in storing and manipulating large amounts of frequently changing number-series and not in supporting and reasoning over infrequently changing data. We have the feeling that database vendors and database users are waiting for each other. As long as vendors do not offer temporal support, the customers are not aware of the potentials of it and as long as the customers do not ask for temporal support, the vendors will not offer it.

After having writen this report, we have the feeling that temporal support does indeed simplify the development and maintenance of applications handling temporal data. As the number of data warehouses (having time as an essential component) increases, the added value of temporal support by the database will probably increase. It seems that up to now, the ignorance of database users of the benefits of temporal support together with the complexity of time theory, prevent the breakthrough of temporal databases.

Recommendations

To get a better idea of the actual added value of temporal support, comparing experiments should be carried out. An interesting comparison would be between an “ordinary” relational database implementation, a temporal implementation and a time series implementation. These experiments will also indicate whether temporal support does indeed offer added value over already existing time series support. If so, this knowledge can be used to convince major database vendors of the benefit of temporal support.

6 References

[ANSI96a] Snodgrass, R.T, Böhlen, M.H et. al.; Adding Transaction Time to SQL/Temporal, ANSI change proposal, ANSI X3H2-96-502r2, October 1996.

[ANSI96b] Snodgrass, R.T, Böhlen, M.H et. al.; Adding Valid Time to SQL/Temporal, ANSI change proposal, ANSI X3H2-96-501r2, October 1996.

[Böhlen95] Böhlen, M.H. Temporal Database System Implementations. SIGMOD Record 24 (4), December 1995

[Böhlen95a] Böhlen, M.H , Announcement of ChronoLog 4.0,

http://www.cs.auc.dk/~boehlen/Software/ChronoLog4.0/ANNOUNCEMENT

[Böhlen96] Böhlen, M.H; Jensen, C.S., *Seamless integration of time into SQL*, submitted to ACM Transactions on Dtabase Systems, December 1996. http://www.cs.auc.dk/~tigeradm/

[Dey96] Dey, D. Temporal Relations and Temporal Normal Forms, Louisiana state university, dep of IS and decision sciences, July 1996

[Etzion98] O. Etzion , S. Jajodia , S. Sripada, *Temporal Databases: Research and Practic,* Lecture notes in computer science, Vol. 1399*,* Springer, pp 115-128, ISBN 3-540-64519-5, 1998

[OS95] Ozsoyoglu, G.; Snodgrass, R.T. Temporal and Real-Time Databases: A Survey. IEEE Transactions for Knowledge and Data Engineering 7 (4). August 1995, pp. 513-532

[SA86] Snodgrass, R.T.; Ahn, I. Temporal Databses. IEEE Computer 19(9), September 1986, pp. 35-42

[Schmidt95] Schmidt, D et. al. Time Series, a Neglected Issue in temporal Database Research?, Proceedings of the Int. workshop on temporal databases, Switzerland, 17-18 September 1995.

[TDBG98] Jensen, C.S.; Clifford, J.; Elmasri, R.; Gadia, S.K.; Hayes, P.; Jajodia, S. (eds.) Dyreson, C.; Grandi, F.; Käfer, W.; Kline, N.; Lorentzos, N.; Mitsopoulos, Y.; Montanari, A.; Nonen, D.; Peressi, E.; Pernici, B.; Roddick, J.F.; Sarda, N.L.; Scalas, M.R.; Segev, A.; Snodgrass, R.T.; Soo, M.D.; Tansel, A.; Tiberio, P.; Wiederhold, G. A Consensus Glossary of Temporal Database Concepts. At URL http://www.cs.auc.dk/~csj/Glossary/

[Tiger98] Böhlen, M, *Tiger reference manual*, Department of computer science, Univerisity of Aalborg, http://www.cs.auc.dk/~tigeradm/

[TSQL94] Snodgrass, R.T; Ahn, I. et. al., TSQL2 language specification, September 1994. At URL http://www.cs.arizona.edu/people/rts/tsql2.html

[Wilson96] Wilson, C.C.V., *Geographic Information Systems and Time*, Department of Geography, Carleton University, Ottawa,Canada, 1996 (http://www.carleton.ca/~cwilson/thesis.html)

[ZCFS+97] Zaniolo, C.; Ceri, S.; Faloutsos, C.; Snodgrass, R.T.; Subrahmanian, V.S.; Zicari, R. Advanced database systems. Morgan Kaufmann Publishers, San Francisco, California, 1997

Appendix A: Temporal query languages

This appendix shows some temporal relational query languages and temporal object oriented query languages that have appeared in the literature (taken from [ZCFS+97]).

A.1 Temporal relational query languages

Name	Underlying data model	based on	formal semantics
HQL	Sadeghi	DEAL	x
-	HDM	Ils	x
Time-By-Example	Tansel	QBE	x
-	Bassiouni	Quel	x
HTQuel	Gadia-1	Quel	x
-	Gadia-2	Quel	
Tquel	Snodgrass	Quel	x
Hquel	Tansel	Quel	x
-	ADM	Relational Algebra	x
-	DM/T	Relational Algebra	x
-	HRDM	Relational Algebra	x
Legol 2.0	Jones	Relational Algebra	
Temporal Relational Algebra	Lorentzos	Relational Algebra	x
-	McKenzie	Relational Algebra	x
TOSQL	Ariav	SQL	
-	Ben-Zvi	SQL	x
TSQL	Navathe	SQL	
HSQL	Sarda	SQL	
TDM	Segev	SQL	x
TempSQL	Yau	SQL	x
TSQL2	BCDM	SQL-92	
IXSQL	Lorentzos	SQL-92	x

A.2 Temporal object oriented query languages

Name	Underlying data model	Based on	Implemented
-	Sciore-1	Annotations	
OODAPLEX	OODAPLEX	DAPLEX	
-	Sciore-2	EXTRA/EXCESS	
VISION	Caruso	Metafunctions	x
OQL/T	OSAM*/T	OSAM*/OQL	
PICQUERY+	TEDM	PICQUERY	x
Postquel	Postgres	Quel	x
MATISSE	MATISSE	SQL	x
OQL	OVM	SQL	x
Orion	Kim	SQL	x
OSQL	IRIS	SQL	x
TOOSQL	TOODM	SQL	x
TQL	TIGUKAT	SQL	x
TMQL	TMAD	SQL	
TOSQL	TOODM	SQL	

Appendix B: Relational and Object-Oriented data models

This appendix shows some temporal relational models and temporal object oriented models that have appeared in the literature (taken from [ZCFS+97]).

B.1 Temporal relational datamodels

Data model name	Temporal dimension(s)	Identifier
Accounting Data Model	Both	ADM
-	Both	Ahn
Temporally Oriented Data Model	Both	Ariav
-	Valid	Bassinouni
-	Both	Bhargava
Bitemporal Conceptual Data Model	Both	BCDM
Time Relational Model	Both	Ben-Zvi
DATA	Transaction	DATA
DM/T	Transaction	DM/T
Extensional Data Model	Both	EDM
Homogeneous Relational Model	Valid	Gadia-1
Heterogeneous Relational Model	Valid	Gadia-2
Historical Data Model	Valid	HDM
Historical Relational Data Model	Valid	HRDM
-	Valid	Jones
-	Transaction	Lomet
Temporal Relational Model	Valid	Lorentzos
-	Valid	Lum
-	Both	McKenzie
Temporal Relational Model	Valid	Navathe
-	Valid	Sadeghi
-	Valid	Sarda
Temporal Data Model	Valid	Segev
-	Both	Snodgrass
-	Valid	Tansel
Time Oriented Databank Model	Valid	Wiederhold
-	Both	Yau

B.2 Temporal object oriented data models

Data model name	Temporal dimension(s)	Identifier	Transaction timestamp representation
-	Both	Caruso	Chronon
IRIS	Transaction	IRIS	Chronon, identifier
-	Transaction	Kim	Version hierarchy
MATISSE	Transaction	MATISSE	Chronon, identifier
OODAPLEX	Arbitrary	OODAPLEX	Arbitrary
OSAM*/T	Valid	OSAM*/T	N/A[20]
OVM	Transaction	OVM	Identifier
Postgres	Transaction	Postgres	Period
-	Arbitrary	Sciore-1	Arbitrary
-	Both	Sciore-2	Chronon
TEDM	Valid	TEDM	N/A
TIGUKAT	Both	TIGUKAT	Identifier
TMAD	Valid	TMAD	N/A
Temporal Object-Oriented Data Model	Both	TOODM	Temporal element

[20] Not Applicable

Appendix C: Temporal database related activities

This appendix gives a description of the European research network CHOROCHRONOS. During the writing of this report we had contact with this group. In addition a list of conferences and workshops is given where temporal database research has been presented.

C.1 CHOROCHRONOS[21]

C.1.1 Objectives

The main objective of CHOROCHRONOS is to allow European researchers working on spatial and temporal databases to achieve a higher understanding of each other's work, integrate their results and methodologies, and advance the state of the art in this area through an intensive three-year research programme. This will culminate in the design and partial implementation of an architecture for Spatiotemporal Database Systems (STDBMS). The Participants will also cooperate, through intensive workshops, with researchers from other disciplines who are dealing with temporal and spatial information in their research, and would benefit from the development of an STDBMS. This network will stimulate training and mobility of young researchers working in the areas of spatial and temporal databases. The Participants will actively pursue dissemination of results throughout European academic institutions and industry.

C.1.2 Participants

- National Tech. Univ. of Athens (NTUA), Computer Science Division, Greece (Project Coordinator) - Prof. Timos Sellis
- Aalborg University (AALBORG), Department of Computer Science, Denmark - Prof. Christian Jensen
- FernUniversität Hagen (HAGEN), Praktische Informatik IV, Germany - Prof. Dr. Ralf Hartmut Güting
- Universita Degli Studi di L'Aquila (UNIVAQ), Dipartimento di Matematica Pura ed Applicata, Italy - Prof. Enrico Nardelli
- Univ. of Manchester - Institute of Science & Technology (UMIST), Department of Computation, United Kingdom - Dr. Manolis Koubarakis and Dr. Babis Theodoulidis
- Politecnico di Milano (POLIMI), Dipartimento di Elettronica e Informazione, Italy - Prof. Barbara Pernici
- Institut National de Recherche en Informatique et en Automation (INRIA), Projet VERSO, France - Dr. Stephane Grumbach and Prof. Michel Scholl
- Aristotle University of Thessaloniki (AUT), Department of Informatics, Greece - Prof. Yannis Manolopoulos and Agricultural
- University of Athens (AUA), Informatics Laboratory, Greece - Prof. Nikos Lorentzos
- Technical University of Vienna (TU VIENNA), Department of Geoinformation, Austria - Prof. Andrew Frank
- Swiss Federal Institute of Technology, Zurich (ETHZ), Institute for Information Systems, Switzerland - Prof. Hans-Jorg Schek

[21] At URL: http://www.dbnet.ece.ntua.gr/~choros/index.html

C.2 Conferences and workshops

- ACM SIGMOD International Conference on Management of Data
- ACM Transactions on Database Systems (TODS)
- ARTDB'97: The Second International Workshop on Active, Real-Time and Temporal Database Systems, Como, Italy.
- CAiSE'97/IFIP 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design
- Computer Software and Applications Conference(COMPSAC'96)
- HERMIS Conference
- ICTL'97: Second International Conference on Temporal Logic, Manchester, England
- IEEE Transactions on Knowledge and Data Engineering (TKDE)
- International Conference on Data Engineering (ICDE)
- International Conference on Database and Expert Systems Applications (DEXA)
- International Conference on Extending database Technology (EDBT)
- International Conference on Temporal Logic
- International Database Engineering and Applications Symposium (IDEAS)
- International Workshop on Active and Real-Time Database Systems. Workshops in Computing
- International Workshop on Temporal Databases, Zurich, Switzerland
- International workshop on temporal reasoning in deductive and object-oriented databases
- International Workshop on Temporal Representation and Reasoning (TIME)
- Spatial and Temporal Reasoning, a AAAI workshop, Seattle, Washington.
- TIME'97, Fourth International Workshop on Temporal Representation and Reasoning, Daytona Beach, Florida
- TRDOOD: First International Post-Conference Workshop on Temporal Reasoning in Deductive and Object-Oriented Databases, Singapore