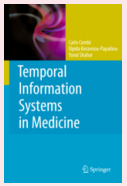
Chapter 3: Temporal Databases[1]

Carlo Combi[1] Elpida Keravnou-Papailiou[2]
Yuval Shahar[3]

[1]Department of Computer Science, University of Verona

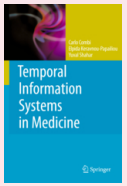
[2]Department of Computer Science, University of Cyprus

[3]Department of Information Systems Engineering, Ben Gurion University

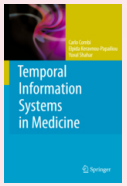


[1]Slides edited with the help of Elena Gaspari, Univ. of Verona

Outline

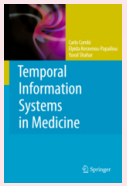


Outline



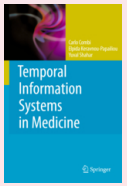
Time and data

- The storage and management of information that has several temporal aspects have been extensively investigated within the research area of *temporal databases*.
  - at the conceptual level: extending the Entity-Relationship data model and the object-oriented data models and the related query languages
  - at the logical level: studying temporal relational and object-oriented data models and languages, and other theoretical topics, such as temporal functional dependencies
  - at the physical level, with the design of access methods to temporal data



Temporal database community

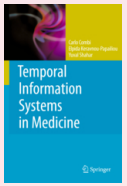
- Definition and update of a “consensus glossary”.
- Proposal of TSQL2: a temporal query language based on the widely used relational query language SQL (Structured Query Language).



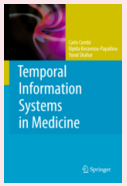
Focus of the chapter

We restrict ourselves to the basic concepts and the main research directions in temporal databases, which have an important role for the modeling and implementation of temporal information systems for medicine.

- Basic features of temporal databases
- Concepts, issues, and solutions proposed for the conceptual and logical modeling and querying of temporal information
- Further important topics as that of modeling and managing multimedia temporal data and semistructured temporal data



Outline

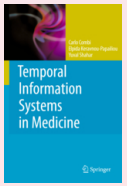


Valid and Transaction Times

Definition

Valid time: the *valid time* (VT) of a fact is the time when the fact is true in the modeled reality.

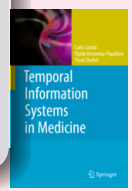
Transaction time: the *transaction time* (TT) of a fact is the time when the fact is current in the database (i.e., it has not been logically deleted).



Valid and Transaction Times: a clinical example (1)

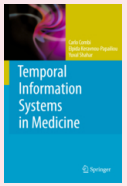
Example

On August 10, 1997, 3:00 p.m., 3:30 p.m., and 4:00 p.m., respectively, three patients convey to the physician their symptom history and the physician enters immediately this data into the database: Mr. Rossi had palpitation symptoms from July 19, 8:30 p.m. to July 20, 1997, 7 a.m., headache from July 1, 8:45 p.m. to July 10, 1997, 1 p.m. and nausea on August 10, 1997, from 10 a.m. to 2 p.m.; Mr. Smith had nausea on June 16, 1997 from 5:30 to 7:45 p.m., and headache on July 5, 1997, from 9:00 a.m. to 5:30 p.m.; Mr. Hubbard had lower extremity edema from May 19, 1997, 5 p.m. to May 21, 9 a.m. By mistake, the physician enters only “edema” instead of “lower extremity edema” for the symptom of Mr. Hubbard.



Valid and Transaction Times: a simple database (1)

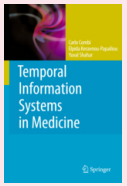
Patient	Symptom	VT	TT
Rossi	palpitation	[97Jul19;8:30, 97Jul20;7:00]	[97Aug10;15:00, ∞)
Rossi	headache	[97Jul1;8:45, 97Jul10;13:00]	[97Aug10;15:00, ∞)
Rossi	nausea	[97Aug10;10:00, 97Aug10;14:00]	[97Aug10;15:00, ∞)
Smith	nausea	[97Jun16;17:30, 97Jun16;19:45]	[97Aug10;15:30, ∞)
Smith	headache	[97Jul5;9:00, 97Jul5;17:30]	[97Aug10;15:30, ∞)
Hubbard	edema	[97May19;17:00, 97May21;9:00]	[97Aug10;16:00, ∞)



Valid and Transaction Times: a clinical example (2)

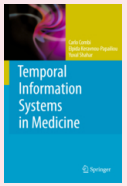
Example

On October 23, 1997, at 4 p.m., Mr. Hubbard goes back to the physician and tells him that he has to declare other symptoms he previously didn't considered as important: chest pain on June 12, 1997 from 8:30 a.m. to 9:15 a.m., and chest and arm discomfort from June 27, 1997 8 a.m. to June 29, 1997 10 p.m. Moreover, the physician corrects the data about the previous symptoms the patient talked about, i.e. the lower extremity edema.



Valid and Transaction Times: a simple database (2)

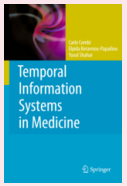
Patient	Symptom	VT	TT
Rossi	palpitation	[97Jul19;8:30, 97Jul20;7:00]	[97Aug10;15:00, ∞)
Rossi	headache	[97Jul1;8:45, 97Jul10;13:00]	[97Aug10;15:00, ∞)
Rossi	nausea	[97Aug10;10:00, 97Aug10;14:00]	[97Aug10;15:00, ∞)
Smith	nausea	[97Jun16;17:30, 97Jun16;19:45]	[97Aug10;15:30, ∞)
Smith	headache	[97Jul5;9:00, 97Jul5;17:30]	[97Aug10;15:30, ∞)
Hubbard	edema	[97May19;17:00, 97May21;9:00]	[97Aug10;16:00, 97Oct23;16:00)
Hubbard	chest pain	[97Jun12;8:30, 97Jun12;9:15]	[97Oct23;16:00, ∞)
Hubbard	chest and arm discomfort	[97Jun27;8:00, 97Jun29;22:00]	[97Oct23;16:00, ∞)
Hubbard	lower extremity edema	[97May19;17:00, 97May21;9:00]	[97Oct23;16:00, ∞)



User-defined Time

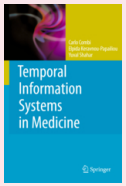
User-defined time is an attribute having time as domain, whose semantics is completely unknown to the system: for example, the birth date of a patient or the date of the first hospital admission of a patient could be two user-defined times.

- User-defined time is not supported by the (temporal) database system.



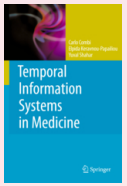
Categories of (temporal) databases

- *Snapshot databases* represent only the current state of the modeled world
- *Valid-time databases*, also known as *historical databases*, support only the valid time
- *Transaction-time databases*, also known as *rollback databases*, support only the transaction time
- *Bitemporal databases* support both transaction and valid times

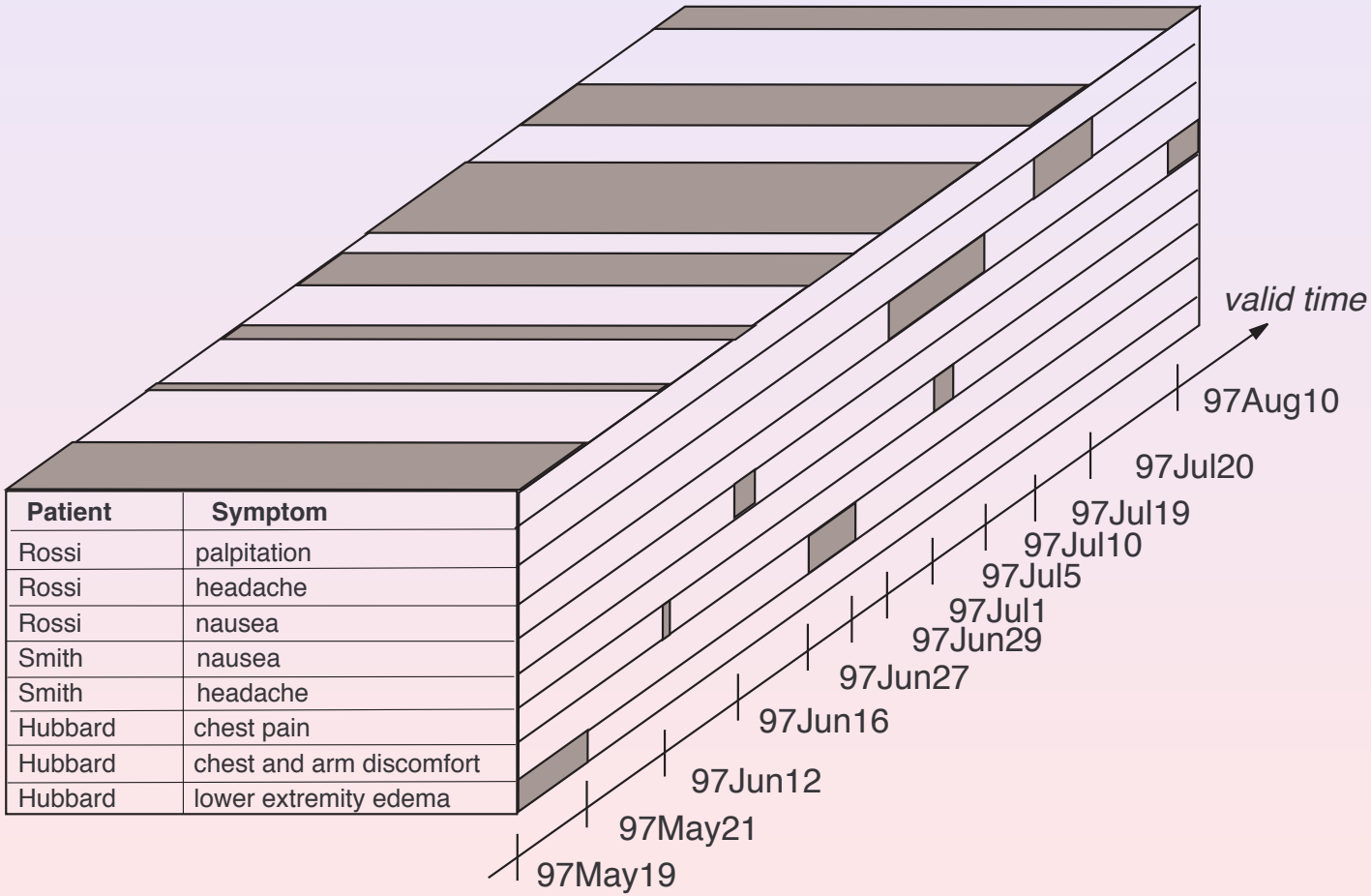


The snapshot database of patients' symptoms

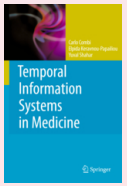
Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea
Smith	nausea
Smith	headache
Hubbard	chest pain
Hubbard	chest and arm discomfort
Hubbard	lower extremity edema



The valid-time database of patients' symptoms



Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea
Smith	nausea
Smith	headache
Hubbard	chest pain
Hubbard	chest and arm discomfort
Hubbard	lower extremity edema



The transaction-time database of patients' symptoms

Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea

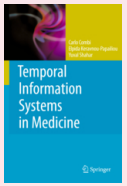
Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea
Smith	nausea
Smith	headache

Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea
Smith	nausea
Smith	headache
Hubbard	edema

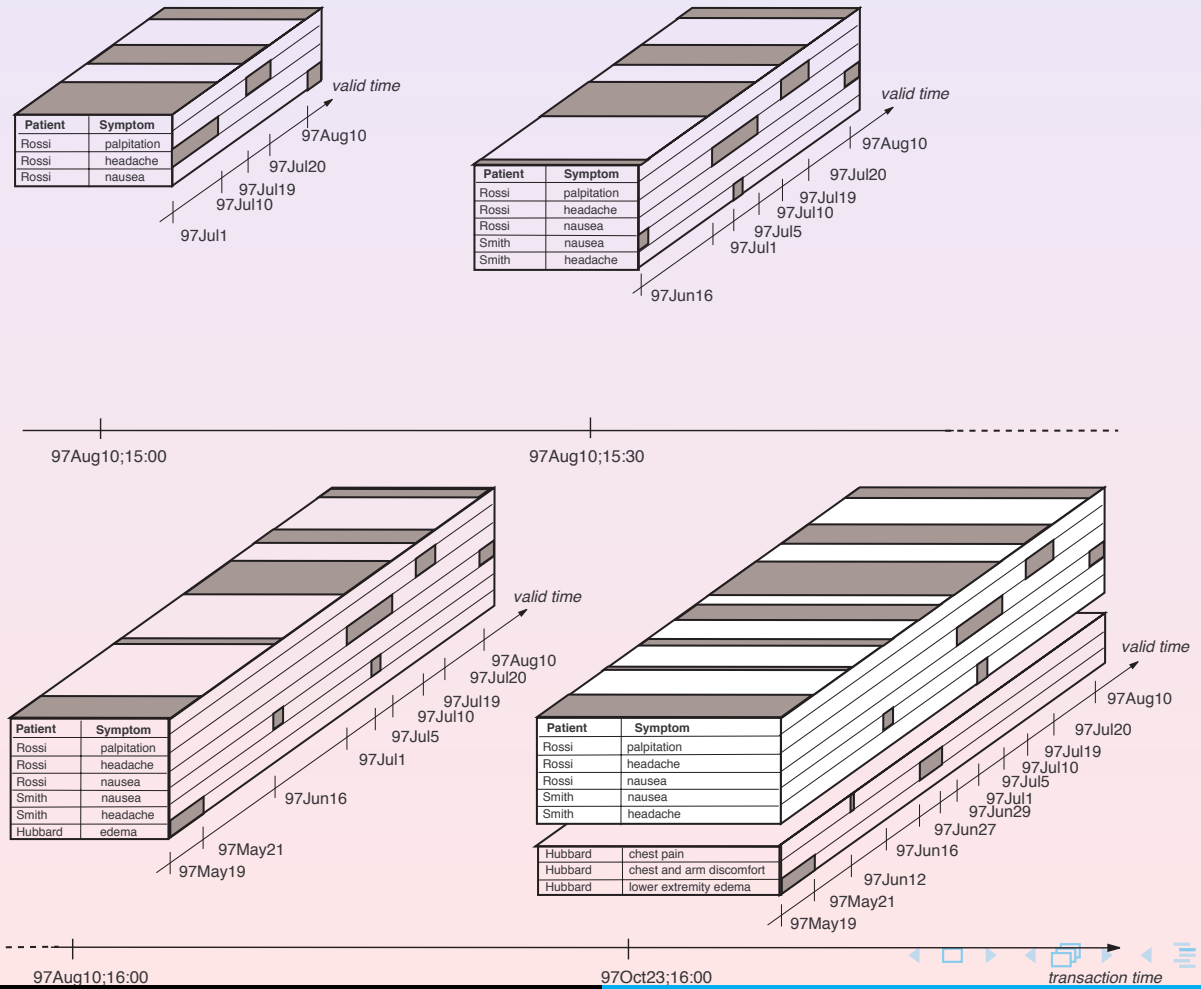
Patient	Symptom
Rossi	palpitation
Rossi	headache
Rossi	nausea
Smith	nausea
Smith	headache

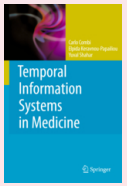
Hubbard	chest pain
Hubbard	chest and arm discomfort
Hubbard	lower extremity edema

97Aug10;15:00 97Aug10;15:30 97Aug10;16:00 97Oct23;16:00 *transaction time*

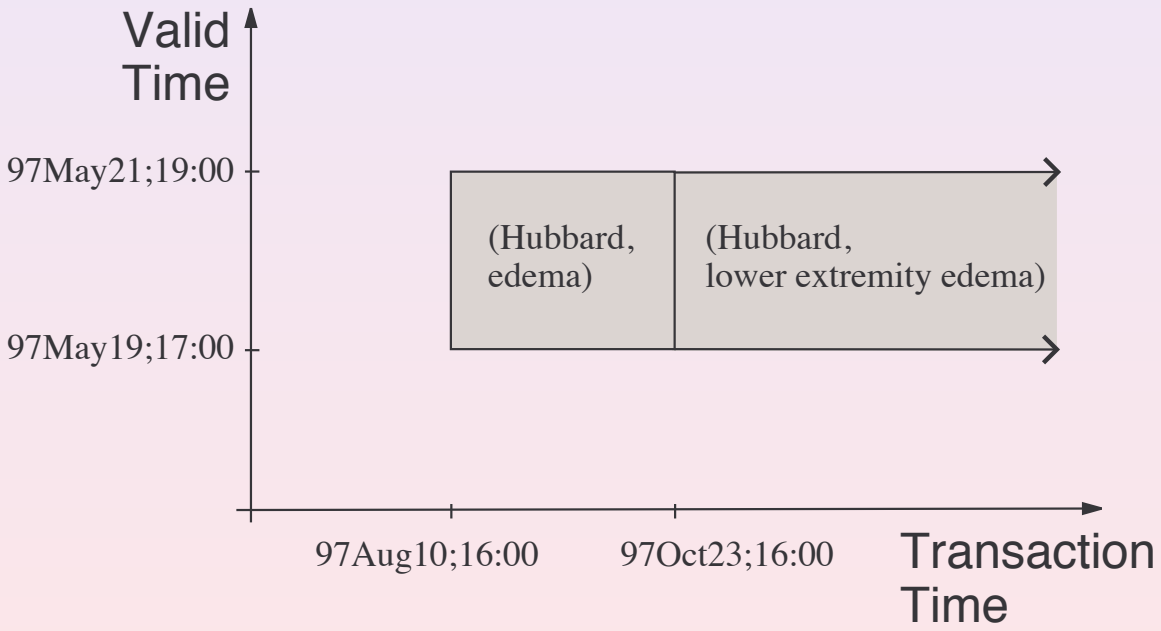


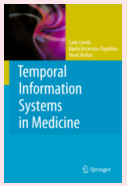
The bitemporal database of patients' symptoms





Graphical representation of tuples according to the bitemporal conceptual model

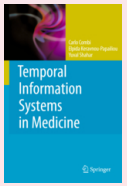




Bitemporal databases: specializations

Valid and transaction times can be related to each other in different ways.
On the basis of the relationships that exist between the starting and/or ending points of the valid and transaction times of each single tuple of a relation, Jensen and Snodgrass characterize classes of relations

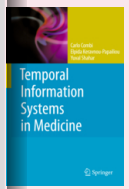
- retroactive, delayed retroactive,
- predictive, early predictive,
- retroactively bounded, strongly retroactively bounded, delayed strongly retroactively bounded,
- general, degenerate, ...



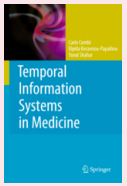
Bitemporal databases: specializations

Example

- Given the starting point VT_s of the valid time and the starting point TT_s of the transaction time, a relation is *retroactive* if, for each tuple of the relation, the relationship $VT_s \leq TT_s$ holds,
- a relation is *delayed retroactive with bound* $\Delta t \geq 0$ if, for each tuple of the relation, the relationship $VT_s \leq TT_s - \Delta t$ holds.
- according to such a criterion, the relation of patient symptoms is *retroactive*, because $VT_s < TT_s$ for every tuple, i.e., the physician enters symptom data after the appearance of the symptoms themselves.



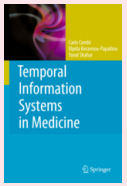
Outline



Relations and Temporal Dimensions

The two main approaches for empowering the relational model with temporal features are based on:

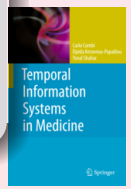
- *Tuple timestamping*. Time dimensions are added at the tuple level.
- *Attribute timestamping*. Time dimensions are added at the attribute level.



A clinical scenario

Example

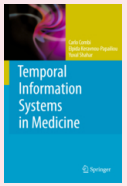
To manage the quality of care at the hospital level, the head of the hospital considers the history of patients' hospitalizations, as maintained in a suitable bitemporal database. At time 4, the administrative operator enters into the database the fact the Mr. Hubbard is hospitalized in the Intensive Care Unit from time 4 to time 9; at time 5, he enters the fact that Mr. Rossi was in the Intensive Care Unit from time 2 to time 3. At time 8, new information is added into the database: Mr. Rossi was in the Cardiology ward from time 1 to time 1 and now he is in the Cardiology ward since time 4; Mr. Smith is in the ward of Internal Medicine from time 5 to time 9 and then he will move to the Neurology ward, where he will stay from time 10 up to a not yet defined time.



A clinical scenario (cont.d)

Example

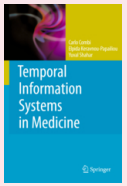
At time 10, the operator enters the (wrong) fact that Mr. Hubbard has been hospitalized in the Cardiology ward since time 10; at time 12 the operator corrects the previous error and enters the correct fact that Mr. Hubbard has been hospitalized in the Pneumology ward since time 10; at time 14, the operator is informed that Mr. Hubbard left the Pneumology ward at time 12, and consequently he corrects data into the database.



Tuple Timestamping and time primitives

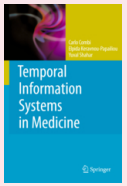
The relation schema has some special attributes, which add (several) temporal dimensions to each tuple of any relation. Different choices can be made about the domain of temporal dimensions.

- time points (chronons);
- time intervals;
- temporal elements.



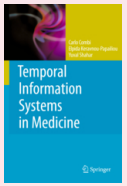
Tuple Timestamping: the (point-based) temporal database

Patient	Ward	VT	TT
Rossi	Cardiology	1	8
Rossi	Cardiology	4	8
Rossi	Intensive Care Unit	2	5
Rossi	null	4	5
Smith	Internal Medicine	5	8
Smith	Neurology	10	8
Hubbard	Intensive Care Unit	4	4
Hubbard	null	10	4
Hubbard	Cardiology	10	10
Hubbard	Pneumology	10	12
Hubbard	null	12	14



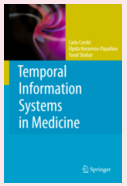
Tuple Timestamping: the (interval-based) temporal database

Patient	Ward	VT	TT
Rossi	Cardiology	[1, 1]	[8, $+\infty$]
Rossi	Cardiology	[4, $+\infty$]	[8, $+\infty$]
Rossi	Intensive Care Unit	[2, 3]	[5, $+\infty$]
Smith	Internal Medicine	[5, 9]	[8, $+\infty$]
Smith	Neurology	[10, $+\infty$]	[8, $+\infty$]
Hubbard	Intensive Care Unit	[4, 9]	[4, $+\infty$]
Hubbard	Cardiology	[10, $+\infty$]	[10, 11]
Hubbard	Pneumology	[10, $+\infty$]	[12, 13]
Hubbard	Pneumology	[10, 12]	[14, $+\infty$]



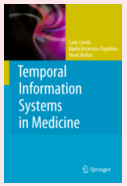
Tuple Timestamping: the (temporal element-based) temporal database

Patient	Ward	VT	TT
Rossi	Cardiology	$[1, 1] \cup [4, +\infty]$	$[8, +\infty]$
Rossi	Intensive Care Unit	$[2, 3]$	$[5, +\infty]$
Smith	Internal Medicine	$[5, 9]$	$[8, +\infty]$
Smith	Neurology	$[10, +\infty]$	$[8, +\infty]$
Hubbard	Intensive Care Unit	$[4, 9]$	$[4, +\infty]$
Hubbard	Cardiology	$[10, +\infty]$	$[10, 11]$
Hubbard	Pneumology	$[10, +\infty]$	$[12, 13]$
Hubbard	Pneumology	$[10, 12]$	$[14, +\infty]$



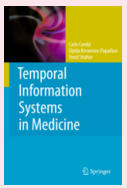
Attribute Timestamping: basic features

- It represents the valid time of each single attribute within a tuple.
- The value of any attribute is complex:
  $$\{Atemporal\ value_1\langle VT_1\rangle, ..., Atemporal\ value_n\langle VT_n\rangle\}$$
- The temporal dimension can be represented by single time points, time intervals, or time elements.
- Some temporal data models distinguish two different categories of attributes, time-varying and constant.
- In other proposals, all the attributes have a temporal dimension: usually each tuple is *homogeneous*, i.e. the time span over which an attribute has (different) value(s) within a tuple is the same for each attribute of the considered tuple.



Attribute timestamping with (bi-dimensional) temporal elements

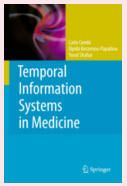
Patient	**Ward**
Rossi $\langle [1, +\infty], [5, +\infty] \rangle$	Cardiology $\langle [1, 1] \cup [4, +\infty], [8, +\infty] \rangle$ Intensive Care Unit $\langle [2, 3], [5, +\infty] \rangle$
Smith $\langle [5, +\infty], [8, +\infty] \rangle$	Internal Medicine $\langle [5, 9], [8, +\infty] \rangle$ Neurology $\langle [10, +\infty], [8, +\infty] \rangle$
Hubbard $\langle [4, 12], [4, +\infty] \rangle$	Intensive Care Unit $\langle [4, 9], [4, +\infty] \rangle$ Cardiology $\langle [10, +\infty], [10, 11] \rangle$ Pneumology $\langle [10, +\infty], [12, 13] \rangle$ Pneumology $\langle [10, 12], [14, +\infty] \rangle$



Temporal algebras

In extending the relational algebra to deal with temporal data, we can identify two different steps:

1. extending the standard (atemporal) relational operators to manage tuples having temporal dimensions;
2. introducing new specific operators for managing explicitly the temporality of tuples.



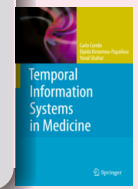
Extending the standard relational operators

- A given temporal algebra has to define the semantics of the usual relational operators (union, intersection, difference, projection, selection, join, cartesian product)

Example

Let us assume that we have to define the *union* operation in a temporal algebra based on tuple-timestamping.

- Informally, we could say that the union of two temporal relations is composed of the tuples of the first relation and those of the second relation.
- What happens for those tuples which have the same values for the atemporal attributes, but different timestamps in the two relations?



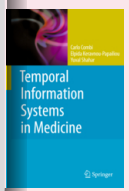
Extending the standard relational operators

- A given temporal algebra has to define the semantics of the usual relational operators (union, intersection, difference, projection, selection, join, cartesian product)

Example

Let us assume that we have to define the *union* operation in a temporal algebra based on tuple-timestamping.

- If the timestamping is provided through temporal elements, then compose a single tuple with the same values of atemporal attributes and with the union of the timestamps of the two tuples as its timestamp.
- If timestamping is through intervals, we have two distinct tuples in the result in case the two timestamps are not intersecting; when the two timestamps of the operands intersect, usually we perform a *coalescing*.

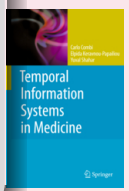


Extending the standard relational operators

Example

Let us consider now the join operation (on atemporal attributes).

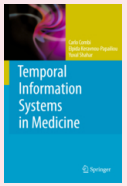
- We should decide whether timestamps have to be considered in joining tuples.
  - A solution could be that we only join tuples with intersecting timestamps.
  - Another solution could allow joining of tuples with disjoint timestamps.
- We have to decide the result of a join on two temporal relations.
- The implicit requirement is that the result is a temporal relation.
- The timestamp could be defined either by default, or by the user, through some suitable parameters specified in the join operation.



Introducing new temporal relational operators

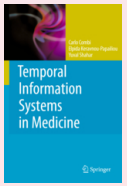
New operators have been proposed in the literature, to deal explicitly with some temporal aspects of data.
Among them, we focus here on the operators

- *timeslice*
- *rollback*
- *nexttuple*



Introducing new temporal relational operators

By the *timeslice* operator we can perform a temporal selection, by considering only those tuples which have the temporal dimension (usually, the valid time) contained in the time interval specified with the operator.

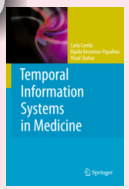


Introducing new temporal relational operators

Example

Let us consider the relation *Pat_Hosp* representing the previously introduced clinical scenario.
We could be interested in considering only those patients, who were hospitalized in the period [4, 10].
According to the standard relational algebra, we can define the following selection:

$$Res\_Pat\_Table \leftarrow \sigma_{(end(VT) \geq 4 \wedge begin(VT) \leq 10)} Pat\_Hosp$$



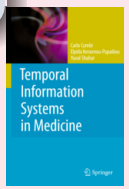
Introducing new temporal relational operators

Although the adoption of operators of standard relational algebra is always possible, the explicit definition of complex selection conditions can be awkward and not convenient for defining complex temporal conditions.

Definition

The *timeslice* operation, expressed by the symbol τ, can be suitably defined as:

$$\tau_t(R) \equiv_{def} \sigma_{(end(VT) \geq begin(t) \wedge begin(VT) \leq end(t))}(R)$$



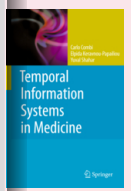
Introducing new temporal relational operators

Example

The database after the valid-time timeslice

$Res\_Pat\_Table \leftarrow \tau_{[4,10]} Pat\_Hosp$

Patient	Ward	VT	TT
Rossi	Cardiology	[4, $+\infty$]	[8, $+\infty$]
Smith	Internal Medicine	[5, 9]	[8, $+\infty$]
Smith	Neurology	[10, $+\infty$]	[8, $+\infty$]
Hubbard	Intensive Care Unit	[4, 9]	[4, $+\infty$]
Hubbard	Cardiology	[10, $+\infty$]	[10, 11]
Hubbard	Pneumology	[10, $+\infty$]	[12, 13]
Hubbard	Pneumology	[10, 12]	[14, $+\infty$]

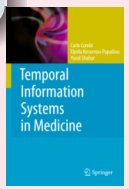


Introducing new temporal relational operators

If needed, transaction time can also be considered in timeslice operation. In this case, the operation is often referred to as *rollback*. The *rollback* operation is usually performed with respect to a time point rather than a time interval. It allows one to represent the state of the database at a specified time point.

Definition

For the *rollback* operation, usually the symbol ρ is adopted:
$\rho_t(R) \equiv_{def} \sigma_{(begin(TT) \leq t \leq end(TT))}(R)$

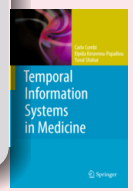


Introducing new temporal relational operators

Example

The database after the *rollback* (transaction-time timeslice)
$Res\_Pat\_Table \leftarrow \rho_9 Pat\_Hosp$

Patient	Ward	VT	TT
Rossi	Cardiology	[1, 1]	[8, $+\infty$]
Rossi	Cardiology	[4, $+\infty$]	[8, $+\infty$]
Rossi	Intensive Care Unit	[2, 3]	[5, $+\infty$]
Smith	Internal Medicine	[5, 9]	[8, $+\infty$]
Smith	Neurology	[10, $+\infty$]	[8, $+\infty$]
Hubbard	Intensive Care Unit	[4, 9]	[4, $+\infty$]



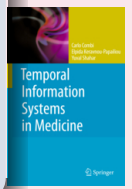
Introducing new temporal relational operators

The operation *nexttuple* allows one to join pairs of successive tuples with respect to the values they assume on the attribute VT.

Definition

Let r be a temporal relation with schema $U \cup \{VT\}$ and let $t, t' \in r$. The application of the operation *nexttuple* to r, denoted $\tau_Z(r)$, with $Z \subseteq U$, joins t, t' if (and only if) $t[Z] = t'[Z]$ and t' is the tuple immediately following t with respect to VT. Formally, $\tau_Z(r)$ is a relation with schema $ZX\overline{X} \cup \{VT, \overline{VT}\}$ defined as follows:

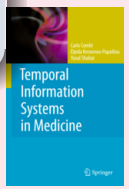
$$\tau_Z(r) \quad \stackrel{\text{def}}{=} \quad p - \pi_{ZX\overline{X} \cup \{VT, \overline{VT}\}}(\sigma_{\overline{VT} > \widetilde{\overline{VT}}}(p \bowtie q)),$$



Introducing new temporal relational operators

Definition

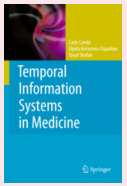
where:
$X = U - Z$,
$p \leftarrow \sigma_{(\overline{VT} > VT)}(r \bowtie \rho_{X,VT \rightarrow \overline{X},\overline{VT}}(r))$,
$q \leftarrow \rho_{Y \rightarrow \widetilde{Y}}(p)$, with $Y = (X\overline{X} \cup \{\overline{VT}\})$.
The resulting temporal relation allows one to associate a tuple t of r with its first evolution t', that is, it associates t with t' if there exists no $t'' \in r$ such that $t''[Z] = t[Z]$ and $t[VT] \leq t''[VT] \leq t'[VT]$.



Temporal relational calculi

Temporal relational calculi extend the standard relational calculus to deal with temporal aspects of tuples.

- the algebra is a *procedural* language, i.e. it allows the user to suitably perform operations in order to reach the desired result;
- the calculus is a *declarative* language, i.e. it allows the user to describe the features of the desired result, without specifying how to reach it.



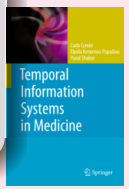
Temporal relational calculi

Similarly to what has been described for temporal algebras, a generic temporal calculus extends the classic (atemporal) relational calculus both by introducing specific operators and functions for the temporal domain, and by considering the temporal dimension of the result.

Example

Let us consider (in a non-standard relational calculus) the expression equivalent to the valid-time timeslice operation $\tau_{[4,10]} Pat\_Hosp$:

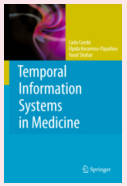
$$\{r \mid (Pat\_Hosp(r) \wedge \exists t(t \in r[VT] \wedge 4 \leq t \leq 10)\}$$



Database languages and temporal dimensions

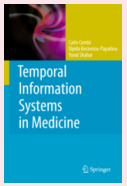
The management of temporal dimensions has an impact on many aspects of database languages:

- statements of the Data Definition Language (DDL), which allow one to express constraints among the different temporal dimensions,
- statements of the Data Manipulation Language (DML), and in particular, of Query Languages (QL), supporting both the specification of temporal search conditions and the definition of the temporal dimensions of the query result.



Database languages and temporal dimensions

- Without loss of generality, we start with the standard `SELECT` command of SQL and introduce, step by step, the specific extensions/modifications needed to deal with temporal dimensions.
- Let a temporal database be defined as a set of *temporal* relations, i.e. relations with both valid and transaction times.

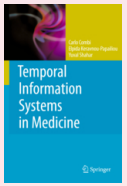


Querying on temporal dimensions

Some basic options:

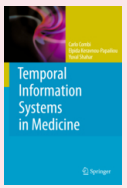
- dealing with temporal and non-temporal data in the `WHERE` clause.
- constraining purely temporal conditions to appear in specific clauses, e.g., `WHEN` and `AS OF`[2].

[2] Obviously, to make it possible to check conditions mixing temporal and non-temporal data, we must allow temporal dimensions to occur, whenever necessary, in the WHERE clause.



Querying on temporal dimensions

- By default, a query is evaluated with respect to the current state of the database, i.e., on the tuples whose transaction time contains the current time (usually identified by the value $+\infty$ or *uc*, for *until changed*).
  - Obviously, there must be the possibility of evaluating the query over both the current and the past states of the database and/or of the information system.
  - This requirement can be managed by adding suitable qualifiers to the `FROM` clause.
- If we are interested in querying the database about its past states, we can use the well-known `AS OF` clause.



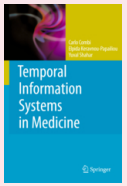
An example database

The relation *pat_sympt*

P_id	symptom	VT	TT
1	headache	[97Oct1, ∞]	[97Oct10, 97Oct14]
2	vertigo	[97Aug8, 97Aug15]	[97Oct15, 97Oct20]
2	vertigo	[97Aug10, 97Aug15]	[97Oct21, ∞]
1	headache	[97Oct1, 97Oct13]	[97Oct15, 97Oct20]
1	headache	[97Oct1, 97Oct14]	[97Oct21, ∞]

The relation *pat_ther*

P_id	therapy	VT	TT
1	aspirin	[96Oct1, 96Oct20]	[97Oct10, ∞]
2	paracetamol	[97Aug11, 97Aug12]	[97Oct15, ∞]



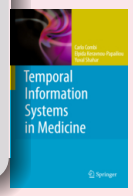
Querying about past states

Example

```
SELECT *
FROM pat_sympt S
AS OF 97Oct20
```

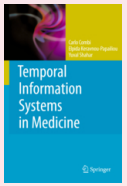
The query returns the content of the relation *pat_sympt* as of October 20, 1997.

P_id	symptom	VT	TT
2	vertigo	[97Aug8, 97Aug15]	[97Oct15, 97Oct20]
1	headache	[97Oct1, 97Oct13]	[97Oct15, 97Oct20]



Further issues

- Dealing with more tables in the `FROM` clause
- Providing some mechanisms to allow the user to obtain a consistent result, that is, a temporal relation endowed with valid and transaction times

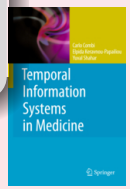


Querying about temporal features

Example

The physician wants to determine the symptoms of patients for which the valid time intervals of symptoms and therapies overlap.

```
SELECT symptom, S.P_id
FROM pat_sympt S, pat_ther T
WHEN NOT(VALID(S) BEFORE VALID(T)) AND
     NOT(VALID(T) BEFORE VALID(S))
WHERE S.P_id = T.P_id
```

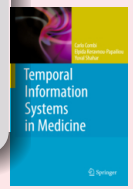


Querying about temporal features

Example

- `BEFORE(·)` is the well-known relation of Allen's Interval Algebra.
- The `WHEN` clause can actually be replaced by a suitable temporal join in the `FROM` clause

```
SELECT symptom, S.P_id
FROM pat_sympt S TJOIN pat_ther T ON
     NOT(VALID(S) BEFORE VALID(T)) AND
     NOT(VALID(T) BEFORE VALID(S))
WHERE S.P_id = T.P_id
```



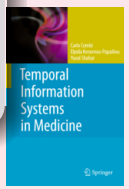
Deriving valid time for query result

- Some default criteria must be specified to determine valid times of the resulting relation, whenever the user does not provide any explicit rule.

Example

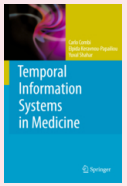
the valid time of each tuple belonging to the resulting relation can be defined as the intersection of the valid times of the corresponding tuples belonging to the relations that occur in the `FROM` clause.

P_id	symptom	VT	TT
2	vertigo	[97Aug11, 97Aug12]	[97Oct21, ∞]



Deriving valid time for query result

- In most cases, the user will explicitly define the way in which the valid times of the resulting relation must be computed, taking into account the meaning he/she assigns to the query and the relative data.

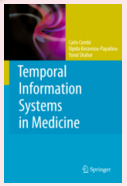


Deriving valid time for query result

Example

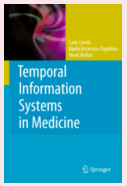
```
SELECT symptom, S.P_id WITH VALID(S) AS VALID
FROM pat_sympt S TJOIN pat_ther T ON
      NOT(VALID(S) BEFORE VALID(T)) AND
      NOT(VALID(T) BEFORE VALID(S))
WHERE S.P_id = T.P_id
```

P_id	symptom	VT	TT
2	vertigo	[97Aug10, 97Aug15]	[97Oct21, ∞]



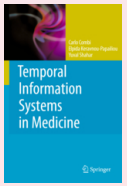
Further temporal clauses

- `TIME SLICE`
- `MOVING WINDOW`



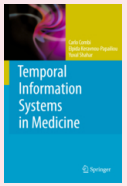
TSQL2

- TSQL2 is the result of the research efforts of several people from the temporal database community, which were grouped in a language definition committee chaired by Richard Snodgrass, one of the pioneers in temporal database research.
- TSQL2 is an extension of the widely accepted standard SQL-92.



TSQL2

- TSQL2 does not add any clause similar to `WHEN`.
- Selection conditions involving valid and transaction times are defined within the `WHERE` clause.
- Valid and transaction times of tuples can be referred to by the `VALID(·)` and `TRANSACTION(·)` constructs.

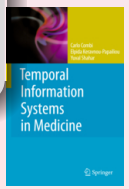


TSQL2

Example

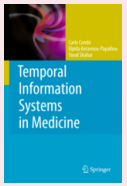
The physician wants to determine the symptoms of patients for which the valid time intervals of symptoms and therapies overlap.

```
SELECT symptom S.P_id
VALID VALID(S)
FROM pat_sympt S pat_ther T
WHERE S.P_id = T.P_id AND
      VALID(S) INTERSECT VALID(T)
```



TSQL2

- TSQL2 allows one to specify, through suitable keywords in the `SELECT` clause, the kind of table required as a result of the query.
- TSQL2 allows the specification of several kinds of relations:
  - *snapshot relations*
  - *valid-time state relations*
  - *valid-time event relations*
  - *transaction-time relations*
  - *bitemporal state relations*
  - *bitemporal event relations*

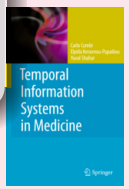


TSQL2

Example

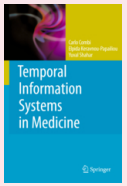
```
CREATE TABLE Hospitalization (Patient
CHAR(30), Ward CHAR(20))
AS VALID STATE DAY AND TRANSACTION
```

- `AS VALID ... AND TRANSACTION` is used to specify the kind of relation the system has to manage.
- The granularity for valid times is that of days, while the granularity for transaction time is system-dependent and therefore is not specified in the statement



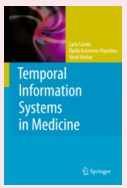
SQL3

- Temporal aspects have also been considered in the development of the most recent ISO standard version of SQL.
- A new component, called SQL/Temporal, was formally approved in July 1995 as part of the SQL3 draft standard and then withdrawn.
- As a matter of fact, most commercial relational database systems provide some extensions to the relational model to represent times and dates through suitable data types and to the SQL query language to allow manipulation of stored time values.



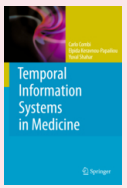
SQL/Temporal

- The SQL3 SQL/Temporal component temporally extends the basic relational model, e.g., by adding the data type `PERIOD` and the two temporal dimensions VT and TT, and it supports temporal queries.
- SQL3 is fully compatible with the previous versions of SQL, so that migration from these atemporal versions is simple and efficient, and existing code can be reused without any additional intervention.
- SQL3 has also been designed to manage temporalities, by providing suitable semantics for the query. Indeed, temporal queries can be specified according to two different semantics, namely, sequenced and non-sequenced semantics.



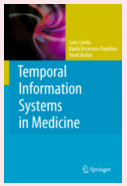
Sequenced and non-sequenced semantics

- According to the *sequenced semantics*, an SQL statement is executed over a given temporal dimension instant by instant, that is, it takes into consideration one time instant (a state of a relation) at a time.
- The *non-sequenced semantics* is more complex than the sequenced one. According to it, an SQL statement accesses a temporal relation as a whole.
  - The non-sequenced semantics for the VT and TT temporal dimensions is imposed by the token `NONSEQUENCED` followed by the specification of the considered temporal dimensions, i.e., `VALIDTIME` and `TRANSACTIONTIME`, respectively.

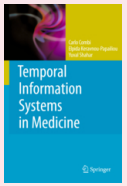


Compatibility

- *Upward compatibility* constrains any new SQL standard to be syntactically and semantically compatible with the existing one. According to this requirement, SQL3 manages atemporal relations, both syntactically and semantically, as SQL does.
- *Temporal upward compatibility* constrains any code available for an atemporal relation to produce the same result if executed over the temporal extension of that relation.

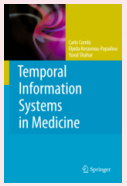


Outline



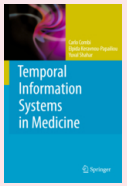
Dealing with time in complex data models

- We now consider in some detail more complex temporal data models, as those based on the Entity-Relationship (ER) data model and on some object-oriented (OO) data models.
- These models allow one to represent complex information, which could not suitably be represented by the relational data model.
- We focus mainly on the valid time of the considered temporal information.



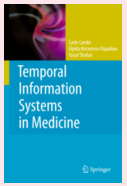
ER basics

- The Entity-Relationship (ER) data model is the main data model adopted for conceptual modeling.
- An *entity* represents any concept, real thing, or object which must be represented in the database application.
- Entities can be associated by *relationships*. A relationship represents any kind of association, we have to represent in the database.
- An entity is characterized by the values of its *attributes*; a relationship is characterized by the entities it associates and (possibly) by the values of some attributes.
  - Attribute values can be atomic, composite, and multivalued.



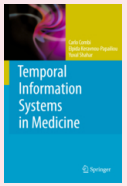
ER basics

- Entities sharing the same features, i.e. the same set of attributes, are described by their *entity type*.
- An entity type is characterized by a name and a set of attributes. Usually, one or more subsets of attributes of a given entity type have a *uniqueness constraint*.
- For each collection of entities of a given entity type, it is not possible to have two or more entities having the same combination of values for the given subsets of attributes. These attributes are called *key attributes*.



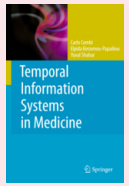
ER basics

- Entity types, called subclasses, can be defined as a refinement (i.e., specialization) of another entity type, which is called its *superclass*.
- Two aspects have to be considered in specialization/generalization.
  - From an *extensional* point of view, entities of a specialized entity type are also members of the set of entities of the corresponding superclass.
  - From an *intensional* point of view, a specialized entity type *inherits* all the features, i.e. the attributes, of its superclass. Several new attributes can be defined in the subclass, to properly specialize it.



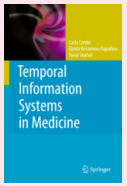
ER basics

- Relationships having similar features are described by *relationship types*
- A relationship type among *n* entity types defines a set of relationships among entities of these types.
- A relationship type is mainly characterized by a name, a set of attributes, a relationship degree, a cardinality ratio (and a participation constraint).
  - The relationship degree is the number of participating entity types.
  - The cardinality ratio specifies the number of relationships an entity of a given entity type can participate in. It is usually expressed by a couple (*min*, *max*).
    - The participation constraint specifies whether at least one relationship for any entity of a given entity type must exist.



ER basics

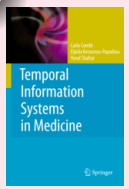
- The ER data model is provided with a powerful graphical notation, to describe the schema of a database.
- ER diagrams provide a graphical representation of the conceptual schema of the considered database, based on the previously introduced concepts of entity and relationship.



ER-based design of clinical database

Example

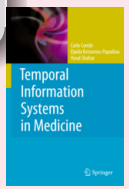
We are required to provide the conceptual design of a database, which allows us to represent information related to follow-up patients. For these patients, usual demographic data is collected (surname, name, birthdate, Address, phone number). The history of patient addresses must be kept by the database, for epidemiological studies. Further data is related to the code, the hospital assigns to a patient, the status of the patient when he was enrolled in the follow-up, and the date when the patient was enrolled. ...



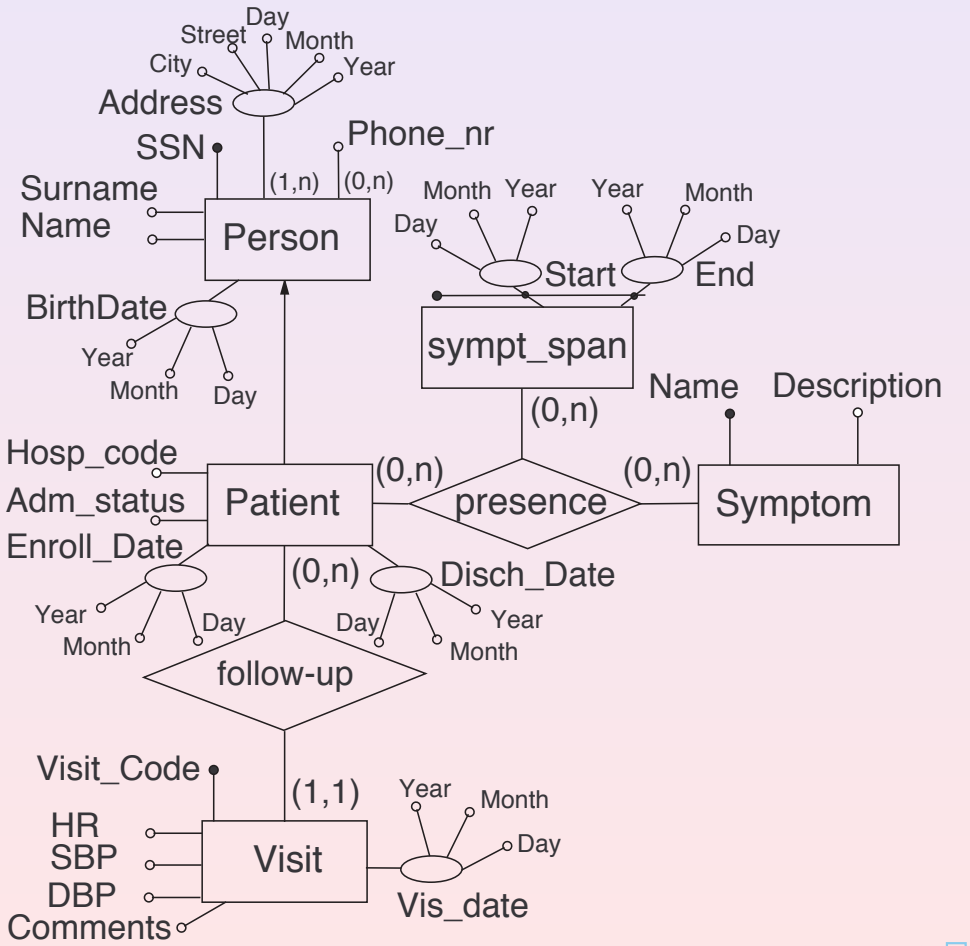
ER-based design of clinical database (cont.d)

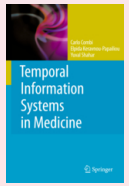
Example

.... At each visit, which is identified by a specific administrative code, data on physiological measurements (Heart Rate, Systolic and Diastolic Pressure) are acquired, together with some comments from the physician, who is visiting the patient. The date at which the visit happens must be recorded. Patient symptoms are recorded for each patient. We need to store the name and a brief description of each symptom. As for temporal aspects of data, the interval during which the patient had a given symptom must be stored in the database.



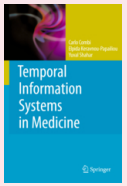
The ER schema for the follow-up patients database





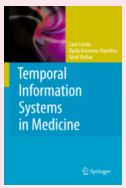
Temporal aspects in the ER schema

- Entity types *Patient* and *Visit* have a temporal dimension, represented by the enrollment and visit dates, respectively: ad-hoc attributes (*Enroll_date* and *Vis_date*, respectively) allow us to represent these temporal aspects.
- To be able to model histories of addresses for patients, we need to associate a temporal dimension to single attributes. In this case, the solution we provide consists of a multivalued composite attribute *Address*. It can have several composite values, composed of *City* and *Street*, for the atemporal part, and *Year*, *Month*, and *Day*, for the temporal dimension.



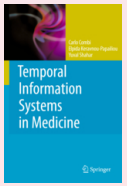
Temporal aspects in the ER schema

- The last issue we have to face, is related to the temporal dimension we have to model for the relationship type *presence*. We need to represent the interval during which a given patient had a given symptom.
- The addition of two suitable composite attributes to the relationship type *presence*, to represent the start and the end of the interval during which the patient had the symptom, seems to be a simple solution, but it is incorrect.



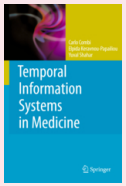
Temporal aspects in the ER schema

- The right solution, then, is to model the interval of appearance of a symptom as an independent entity; the entity type *Sympt_span* has two attributes, collectively key attributes, which represent the start and the end of an interval.
- The entity type *Sympt_span* is related to the entity types *Symptom* and *Patient* by the relationship type *presence*, which becomes a ternary relationship type.



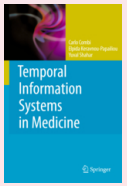
Temporalities and the ER model

- the ER diagrams are difficult to read and to correctly understand, because temporal dimensions of data require the insertion of additional entities, attributes, and relationships, to be represented;
- application-dependent concepts, represented by entities, relationships and attributes, are merged in the diagram with more general concepts, related to temporal aspects.



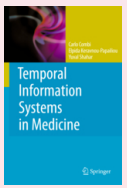
Temporal ER models: pros

- Modeling of temporal information for a given application is simpler and more intuitive;
- entity types of the database schema are designed separately from the more general time-related concepts;
- general temporal aspects of information are directly managed within the data model.

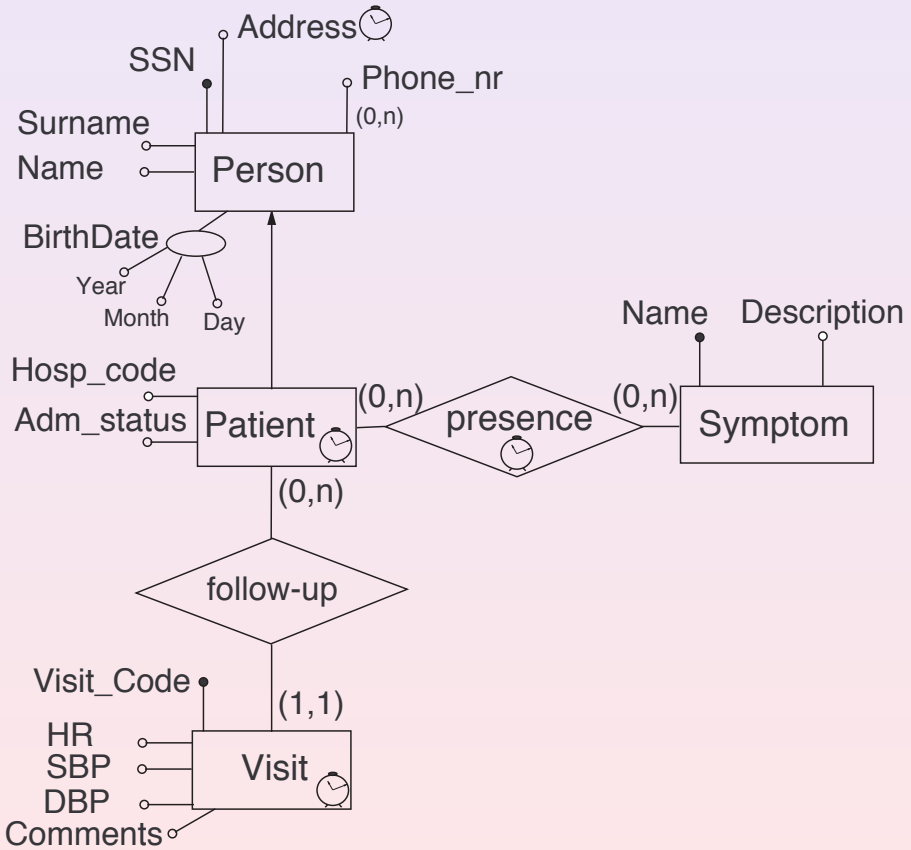


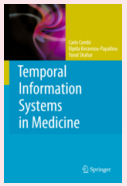
Temporal ER models: cons

- The main disadvantage in adopting a *temporal ER data model* is that the number of basic concepts of the model increases.
  - Some temporal extensions of the ER data model, provide each construct of the model (i.e., entity types, relationship types, attributes) with a "temporal" semantics.
  - If not every model construct is temporal, another consideration is where to add temporal dimensions. Attention has to be paid to the various temporal constraints existing among different entities and/or relationships.



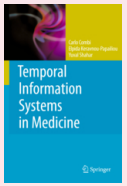
The temporal ER schema for the follow-up patient database





Temporal constraints in the temporal ER schema

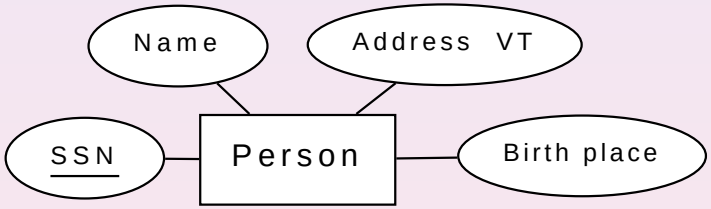
- Intuitively, a constraint must hold for each entity of type *Patient* and the corresponding relationships of type *presence*; the valid time of the relationship must be during the valid time of the entity, because it is not meaningful that a patient had a symptom before starting to be a patient.
- It is worth noting that the definition of the valid time of entities of type *Patient* is application-dependent: the valid time start could be set either by the patient's birthdate or by the enrollment date of the patient.



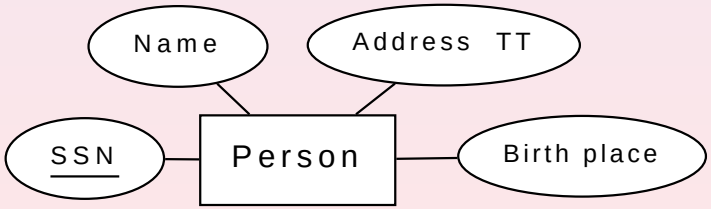
Temporal constraints and temporal ER models

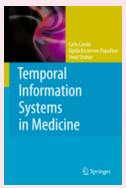
Temporal ER models allows one to capture temporal aspects of information

The address of a person is "via Roma 30, Verona Italy" since December 7, 1998 (valid time)



On September 16, 2008 the address of a person stored into the database is "via Roma 30, Verona Italy" (transaction time)





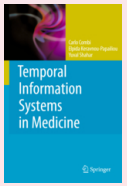
Temporal constraints and temporal ER models

...but what about advanced temporal constraints?

- *The name of a person cannot vary over time*
- *The social security number of a person cannot be reused for a different person in a different time*

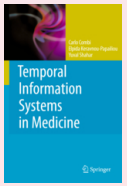
Focus

- Present TimeER with new notations for the specification of advanced temporal constraints, to enhance the expressiveness of the model.
- Define the associated semantics for the new temporal constraints



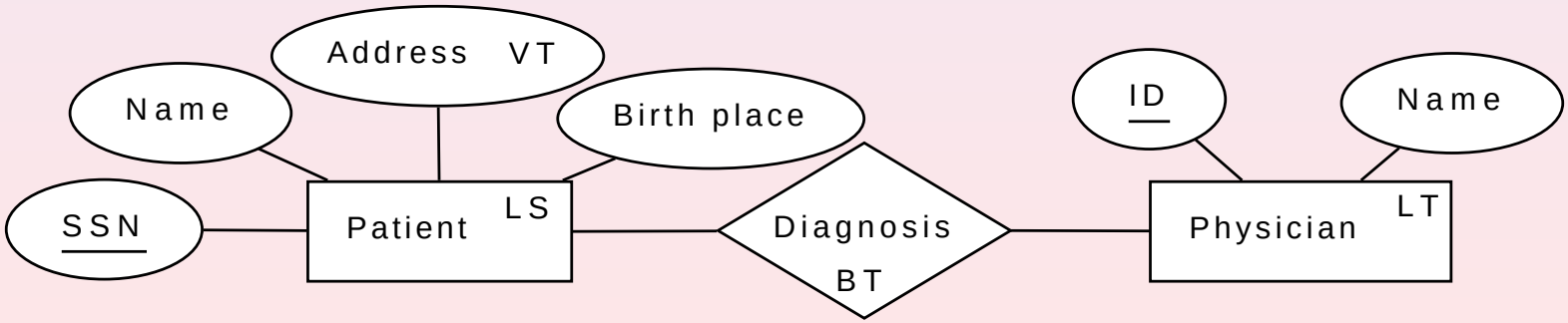
TimeER

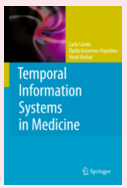
- Extends the EER model by Elmasri and Navathe
- Retains all the existing ER constructs
- Offers new constructs which support the management of temporal aspects of information: *valid time*, *transaction time*, *lifespan*



TimeER

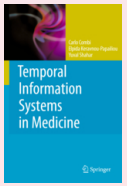
- Extends the EER model by Elmasri and Navathe
- Retains all the existing ER constructs
- Offers new constructs which support the management of temporal aspects of information: *valid time*, *transaction time*, *lifespan*



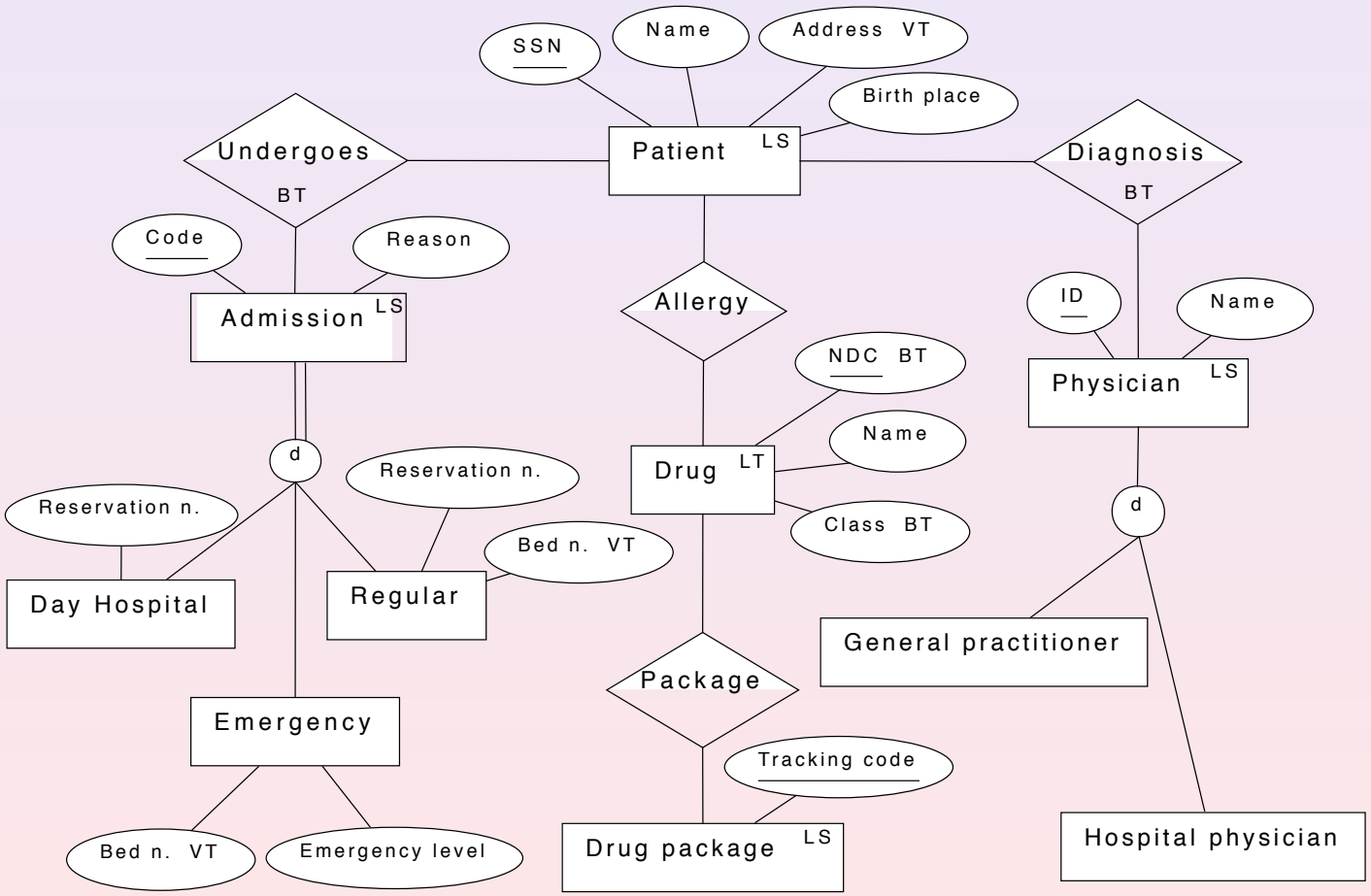


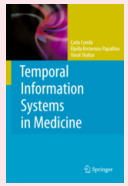
Association of aspects of time to TimeER database concepts

	Entity types	**Relationship types**	**Super/subclass relationships**	**Attributes**
Lifespan	Yes	Yes (entity view)	No	No
Valid time	No	Yes (attribute view)	No	Yes
Transaction time	Yes	Yes	No	Yes



Motivating Example

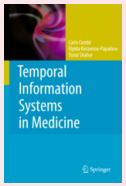




Requirements

- The tracking code of a drug package cannot vary over time, but it can be reassigned to other drug packages (after at least six years from its assignment)
- The NDC (National Drug Code) of a drug can neither vary over time, nor be re-assigned to a different drug
- Birth place cannot vary for a person
- Once a drug allergy is recognized, it cannot disappear

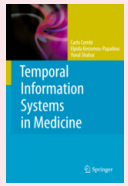
$\longrightarrow$ These constraints cannot be modeled in (the basic) TimeER!



Requirements

- The tracking code of a drug package cannot vary over time, but it can be reassigned to other drug packages (after at least six years from its assignment)
- The NDC (National Drug Code) of a drug can neither vary over time, nor be re-assigned to a different drug
- Birth place cannot vary for a person
- Once a drug allergy is recognized, it cannot disappear

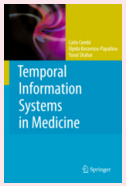
$\longrightarrow$ These constraints cannot be modeled in (the basic) TimeER!



Requirements

- The tracking code of a drug package cannot vary over time, but it can be reassigned to other drug packages (after at least six years from its assignment)
- The NDC (National Drug Code) of a drug can neither vary over time, nor be re-assigned to a different drug
- Birth place cannot vary for a person
- Once a drug allergy is recognized, it cannot disappear

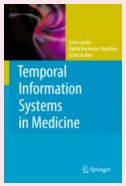
$\longrightarrow$ These constraints cannot be modeled in (the basic) TimeER!



Requirements

- The tracking code of a drug package cannot vary over time, but it can be reassigned to other drug packages (after at least six years from its assignment)
- The NDC (National Drug Code) of a drug can neither vary over time, nor be re-assigned to a different drug
- Birth place cannot vary for a person
- Once a drug allergy is recognized, it cannot disappear

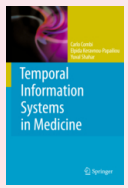
$\longrightarrow$ These constraints cannot be modeled in (the basic) TimeER!



Requirements

- The tracking code of a drug package cannot vary over time, but it can be reassigned to other drug packages (after at least six years from its assignment)
- The NDC (National Drug Code) of a drug can neither vary over time, nor be re-assigned to a different drug
- Birth place cannot vary for a person
- Once a drug allergy is recognized, it cannot disappear

$\longrightarrow$ These constraints cannot be modeled in (the basic) TimeER!

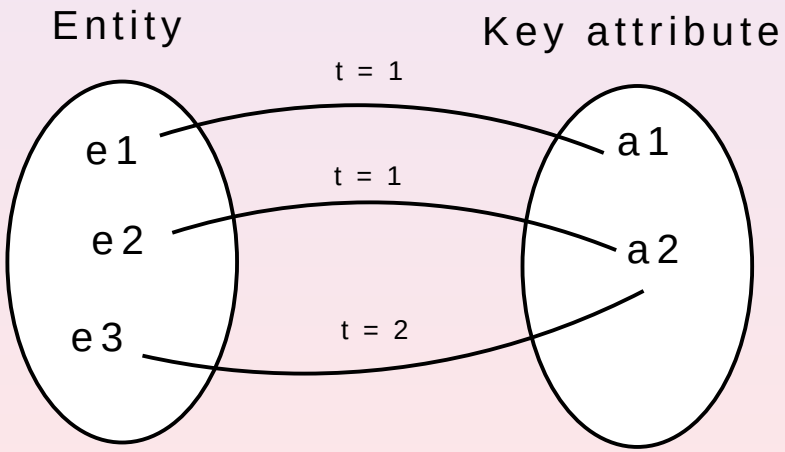


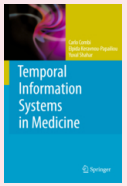
Key Constraints

TimeER key constraints have a **snapshot-reducible** semantics: *at any point in time, the mapping from the entity set to the corresponding set of groups of values for the key attributes is one-to-one.*

Observation 1

The same key value may identify two different entities at two different time instants



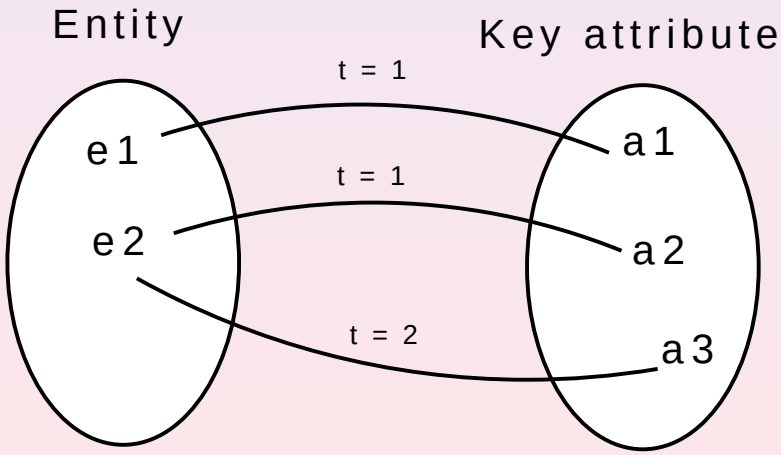


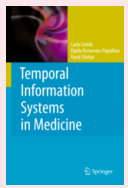
Key Constraints

TimeER key constraints have a **snapshot-reducible** semantics: *at any point in time, the mapping from the entity set to the corresponding set of groups of values for the key attributes is one-to-one.*

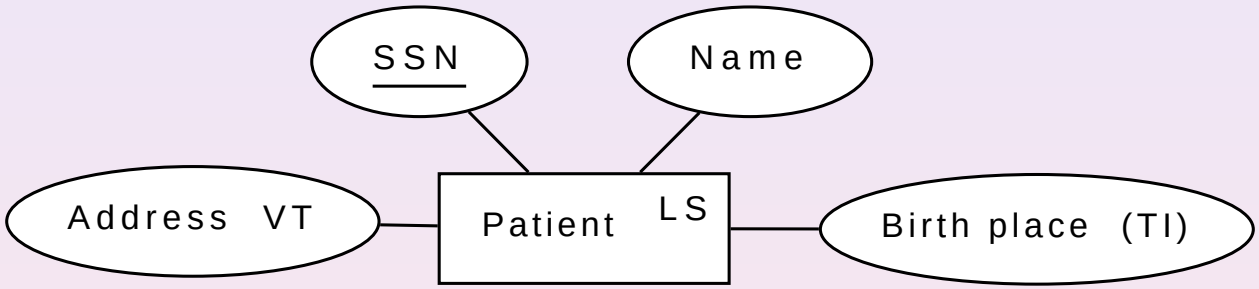
Observation 2

The same entity may have two different key values at two different time instants

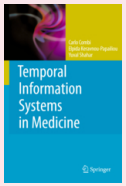




Snapshot-reducible Key



- The SSN (Social Security Number) of a patient can be updated over time



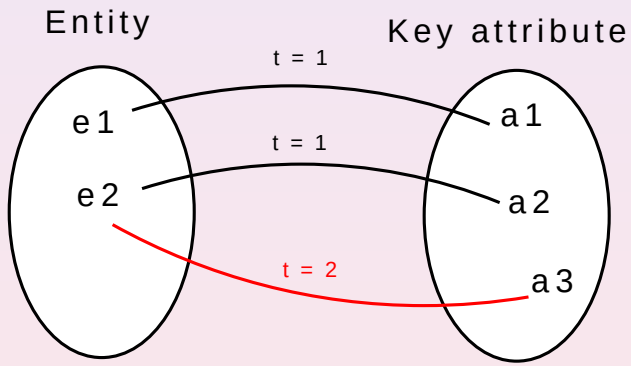
Time-invariant Key

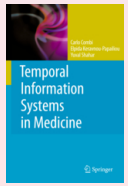
Definition

A *time-invariant key* is a set of attributes of an entity type such that:

- it is a snapshot-reducible key;
- an entity cannot have two different key values in two different valid-time instants.

NB: the same key value may identify two different entities in two different time instants.





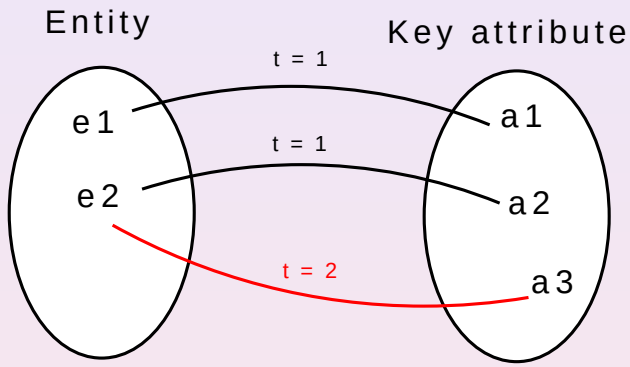
Time-invariant Key

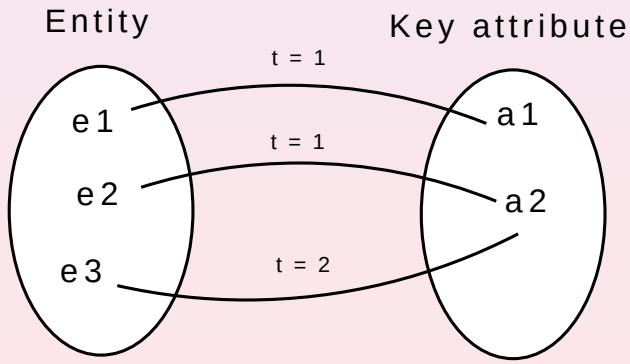
Definition

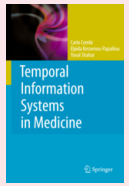
A *time-invariant key* is a set of attributes of an entity type such that:

- it is a snapshot-reducible key;
- an entity cannot have two different key values in two different valid-time instants.

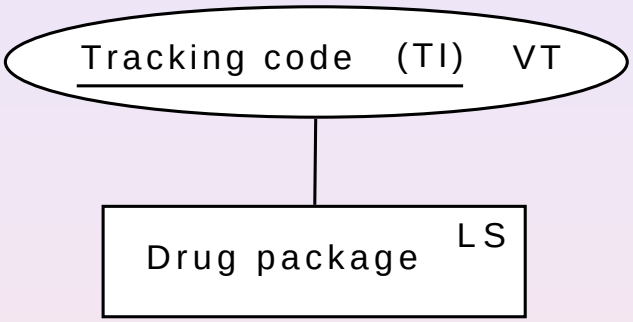
NB: the same key value may identify two different entities in two different time instants.



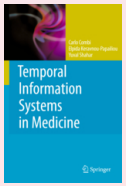




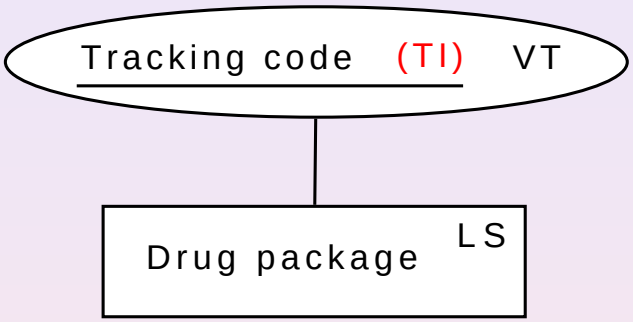
Time-invariant Key



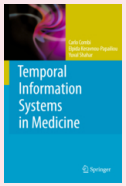
- Two drug packages identified by different tracking codes are different packages
- The same tracking code could identify two different drug packages at two different VT instants



Time-invariant Key



- Two drug packages identified by different tracking codes are different packages
- The same tracking code could identify two different drug packages at two different VT instants

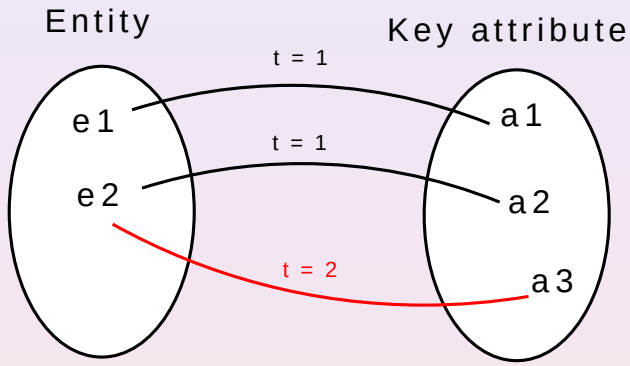


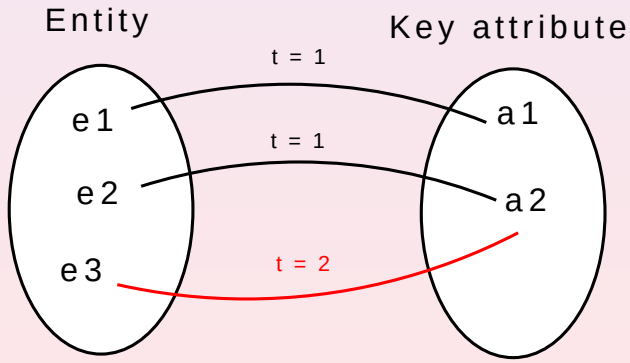
Temporal Key

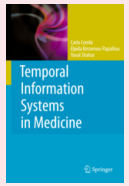
Definition

A *temporal key* is a set of attributes of an entity types such that:

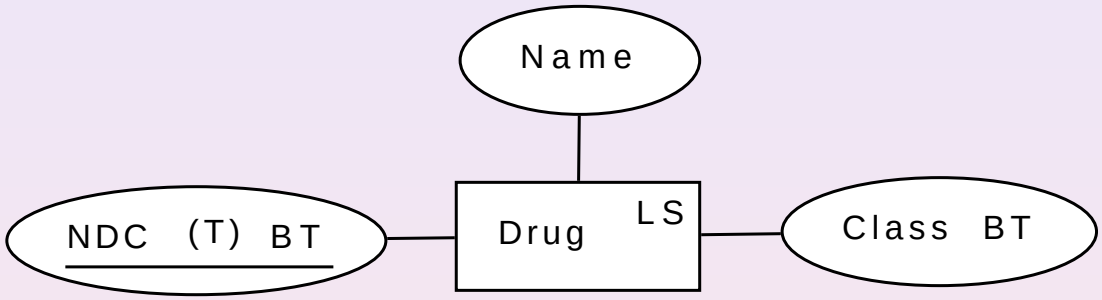
- it is a time-invariant key;
- the same key value cannot identify two different entities in two different valid time instants.



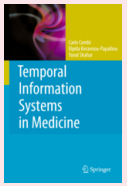




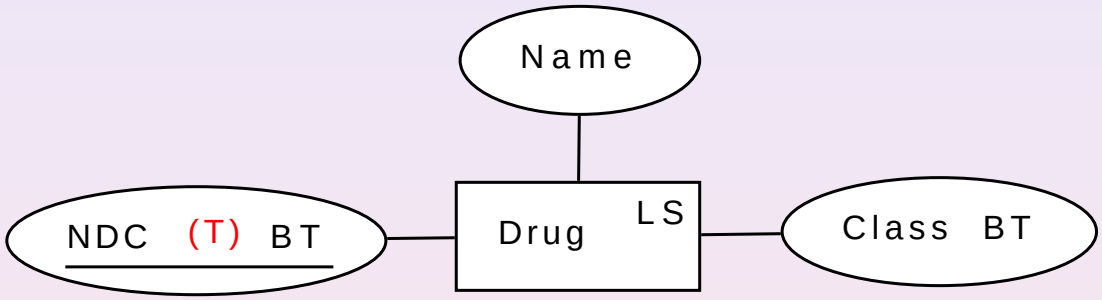
Temporal Key



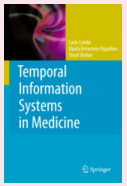
- The NDC of a drug cannot vary over time
- The NDC of a drug cannot be reassigned to a different drug



Temporal Key



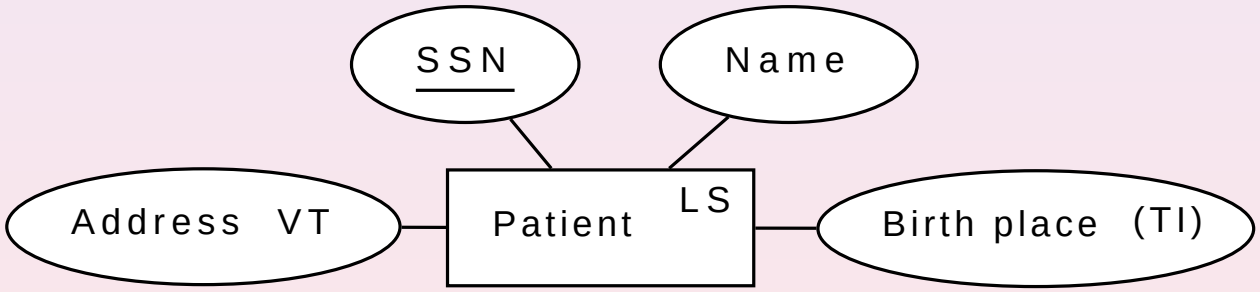
- The NDC of a drug cannot vary over time
- The NDC of a drug cannot be reassigned to a different drug



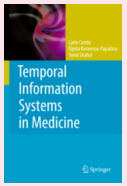
Time-invariant Attribute

Definition

A *time-invariant attribute* is an attribute whose value does not change over time in the valid-time domain.



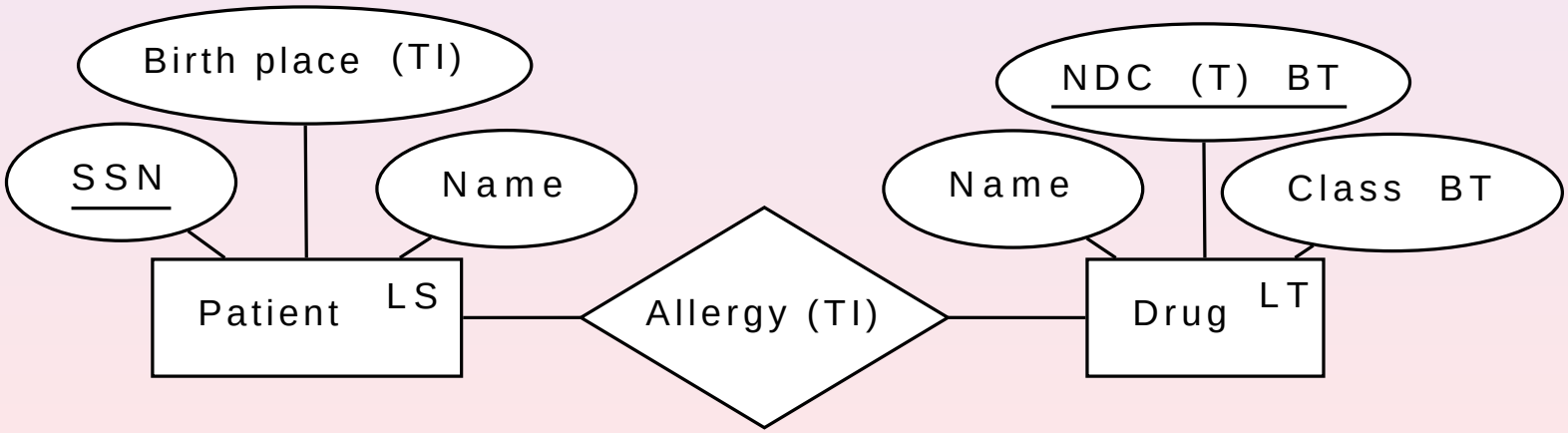
- The birth date of a patient cannot change over time

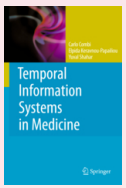


Time-invariant Relationship

Definition

A relationship between two entities is **time-invariant** if, once it has been established, it holds as long as both involved entities exist in the mini-world.

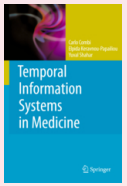




Superclass/Subclass Participation Constraints

Definition (Temporally-Total Superclass/Subclass Relationship)

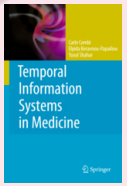
Let E and $E_1, \ldots, E_n$ be TimeER entities such that E is a superclass and $E_1, \ldots, E_n$ are subclasses of E. If the superclass/subclass relationship is *temporally total*, then each member of the superclass is a member of at least one of the subclasses for at least one time instant in its lifespan.



Superclass/Subclass Participation Constraints (cont.d)

Definition (Temporally-Disjoint Superclass/Subclass Relationship)

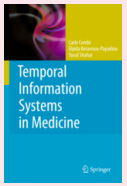
Let E and $E_1, \ldots, E_n$ be TimeER entities such that E is a superclass and $E_1, \ldots, E_n$ are subclasses of E. If the superclass/subclass relationship is *temporally disjoint* then an instance e of E is a member of at most one of the subclasses for all times in its lifespan.



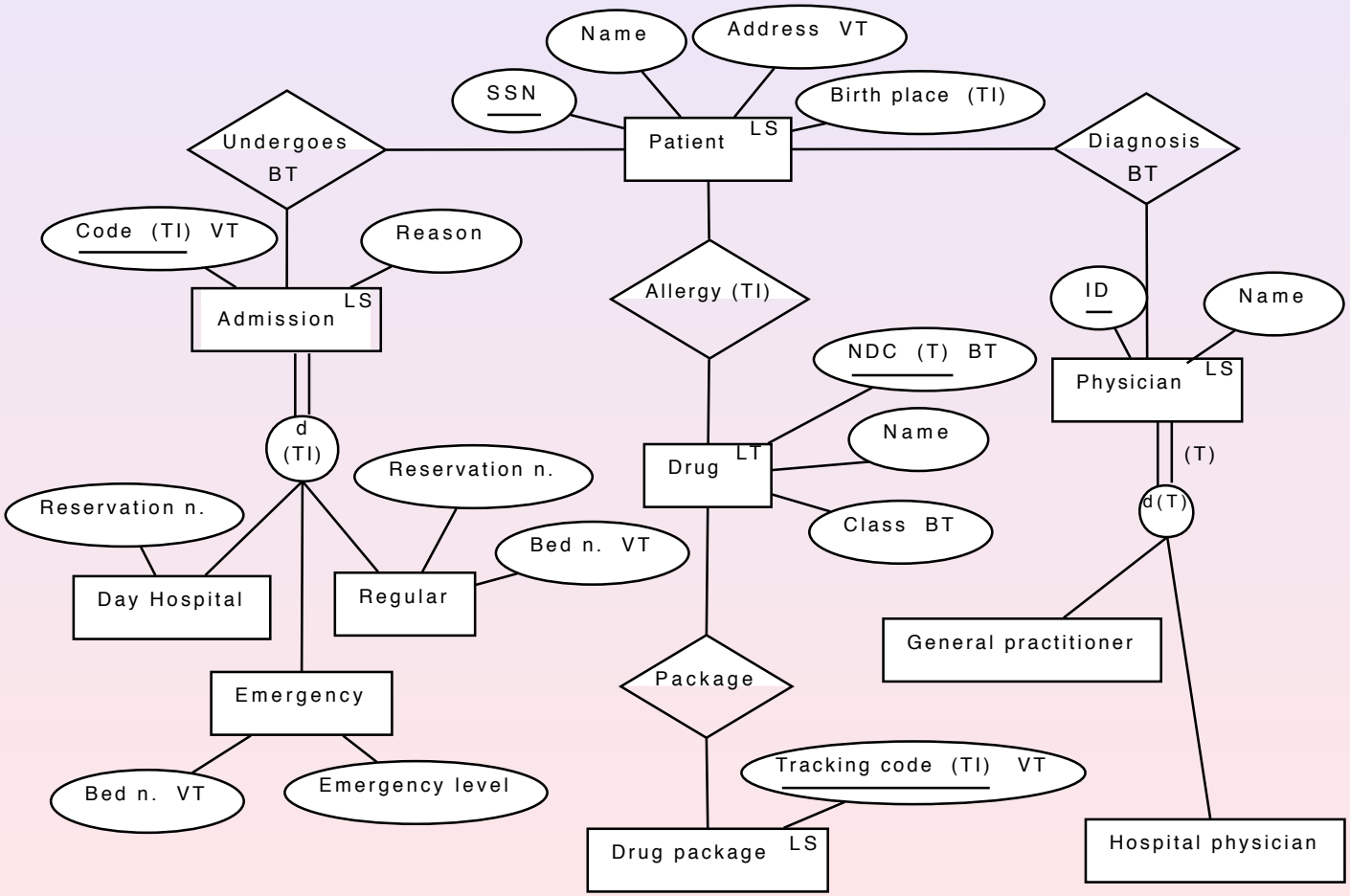
Superclass/Subclass Participation Constraints (cont.d)

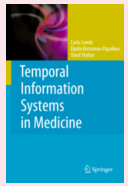
Definition (Time-Invariant Superclass/Subclass Relationship)

A superclass/subclass relationship is *time-invariant* if each member of the superclass that belongs to one or more subclasses is a member of those subclasses for all of its lifespan.



Motivating Example with New Constraints

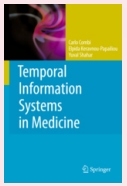




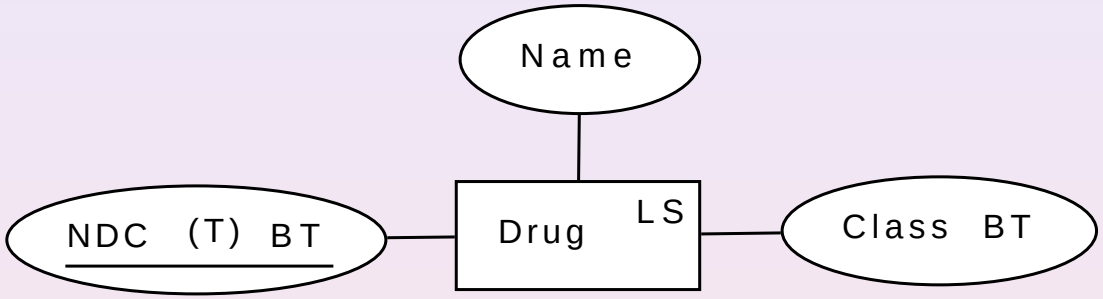
Semantics

Mapping of TimeER diagrams to the *surrogate-based relational model*

- Surrogate: system-generated unique identifier, that cannot be modified nor be reassigned
- Two kinds of relations:
  - *E-relations*: represent TimeER entity types and relationship types that are considered to exist in their own right; has a single E-attribute (surrogate) and a number of time attributes
  - *A-relations*: represent entity or relationship attributes; they reference, through a surrogate, the corresponding E-relation.



Semantics



Drug

drugø	$\overline{LS_s}$	LS_e

Drug_NDC

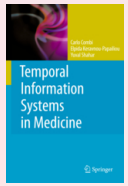
drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e

Drug_name

drugø f.k.	name

Drug_class

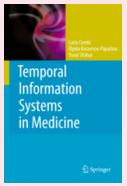
drugø f.k.	class	VT_s	VT_e	TT_s	TT_e



Semantics - Temporal Key Constraint

- Let E be a TimeER entity with a temporal key for which both VT and TT are captured.
- Let R be the A-relation storing the temporal key of E, and let r_i, r_j be tuple variables over R.
- Let X be the group of attributes of R that represents the temporal key of E.
- Let $R\emptyset$ be the foreign key of R referring to the surrogate attribute of the E-relation representing E. Then:

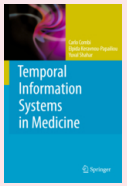
$$\forall r_i, r_j \in R\ (((r_i.R\emptyset = r_j.R\emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Leftrightarrow r_i.X = r_j.X) \wedge ((r_i.X = r_j.X \wedge [r_i.VT_s, r_i.VT_e] \cap [r_j.VT_s, r_j.VT_e] \neq \emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Rightarrow r_i = r_j))$$



Semantics - Temporal Key Constraint

- Let E be a TimeER entity with a temporal key for which both VT and TT are captured.
- Let R be the A-relation storing the temporal key of E, and let r_i, r_j be tuple variables over R.
- Let X be the group of attributes of R that represents the temporal key of E.
- Let $R\emptyset$ be the foreign key of R referring to the surrogate attribute of the E-relation representing E. Then:

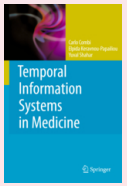
$$\forall r_i, r_j \in R\ (((r_i.R\emptyset = r_j.R\emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Leftrightarrow r_i.X = r_j.X) \wedge ((r_i.X = r_j.X \wedge [r_i.VT_s, r_i.VT_e] \cap [r_j.VT_s, r_j.VT_e] \neq \emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Rightarrow r_i = r_j))$$



Semantics - Temporal Key Constraint

- Let E be a TimeER entity with a temporal key for which both VT and TT are captured.
- Let R be the A-relation storing the temporal key of E, and let r_i, r_j be tuple variables over R.
- Let X be the group of attributes of R that represents the temporal key of E.
- Let $R\emptyset$ be the foreign key of R referring to the surrogate attribute of the E-relation representing E. Then:

$$\forall r_i, r_j \in R\ (((r_i.R\emptyset = r_j.R\emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Leftrightarrow r_i.X = r_j.X) \wedge ((r_i.X = r_j.X \wedge [r_i.VT_s, r_i.VT_e] \cap [r_j.VT_s, r_j.VT_e] \neq \emptyset \wedge [r_i.TT_s, r_i.TT_e] \cap [r_j.TT_s, r_j.TT_e] \neq \emptyset) \Rightarrow r_i = r_j))$$



Semantics - Examples

- Satisfaction of the temporal key constraint

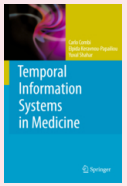
Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	NOW	1	10
ø1	60575-4112-01	1	NOW	11	UC
ø2	00002-7597-01	1	NOW	1	UC

- Violation of the temporal key constraint

Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	10	1	UC
ø1	60575-4112-01	11	NOW	11	UC
ø2	00002-7597-01	1	20	1	UC
ø3	00002-7597-01	21	NOW	21	UC



Semantics - Examples

- Satisfaction of the temporal key constraint

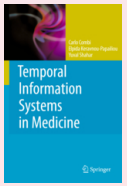
Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	NOW	1	10
ø1	60575-4112-01	1	NOW	11	UC
ø2	00002-7597-01	1	NOW	1	UC

- Violation of the temporal key constraint

Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	10	1	UC
ø1	60575-4112-01	11	NOW	11	UC
ø2	00002-7597-01	1	20	1	UC
ø3	00002-7597-01	21	NOW	21	UC



Semantics - Examples

- Satisfaction of the temporal key constraint

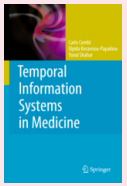
Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	NOW	1	10
ø1	60575-4112-01	1	NOW	11	UC
ø2	00002-7597-01	1	NOW	1	UC

- Violation of the temporal key constraint

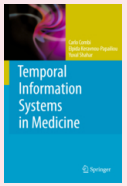
Drug_NDC

drugø f.k.	NDC u.k.	VT_s	VT_e	TT_s	TT_e
ø1	50242-0040-62	1	10	1	UC
ø1	60575-4112-01	11	NOW	11	UC
ø2	00002-7597-01	1	20	1	UC
ø3	00002-7597-01	21	NOW	21	UC



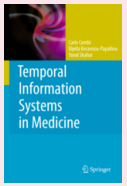
OO methodologies and technologies

- Object-oriented technology applied to the database field has some useful features - abstract data type definition, inheritance, complex object management - in modeling and managing complex information, as that related to clinical medicine.
- Object-oriented design methodologies and standards have received much attention in the last decade:
  - the Object Modeling Technique (OMT) methodology and the related Unified Modeling Language (UML),
  - database-oriented Object Database Management Group (ODMG) standard.



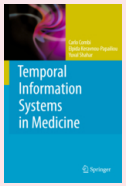
OO basic concepts

- An *object* can model any entity of the real world, e.g., a patient, a therapy, a time-interval.
  - *identity*: provided by the database system through an identifier, called OID
  - *state*: described by properties (*attributes* and *relationships* with other objects) not accessible from the outside
  - *behavior*: defined by *methods*, describing modalities, by which it is possible to interact with the object itself.



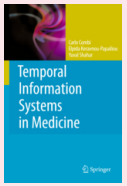
OO basic concepts

- Objects are created as instances of a *class*.
- A class (named also *type* in some proposals) describes
  - the structure of properties through attributes and relationships
  - the behavior of its instances through methods applicable to objects, i.e. instances of the class.
- The *extent* of a class is the set of all objects of that class stored into the database at a given moment.

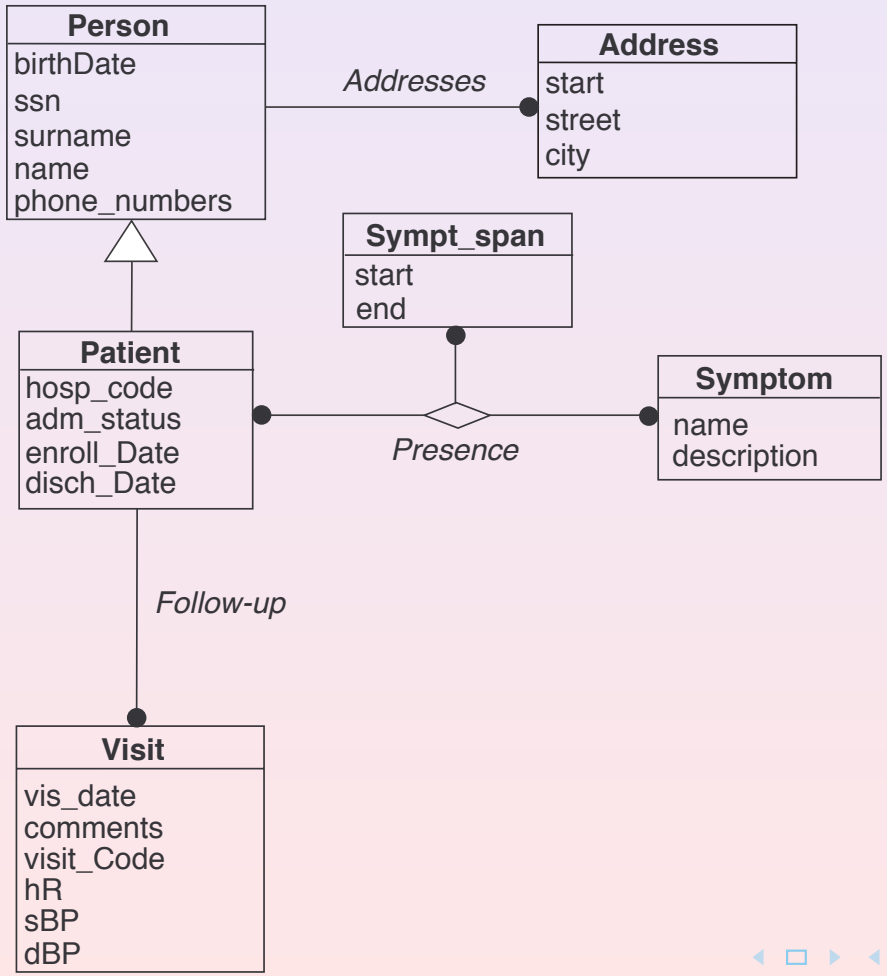


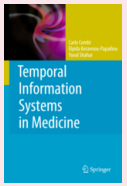
OO basic concepts

- Each method has a declaration (also called *signature*), consisting of a name, a set of parameters, identified by name and their class, and a result, identified by its class.
- Like attributes, code associated to the execution of a method is not accessible from outside.
- This feature is called, in the object-oriented literature, *encapsulation*.
- Further features of object oriented databases are inheritance, polymorphism, management of complex objects, and persistence.



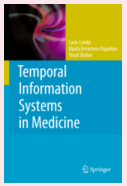
Object-oriented schema of the database about follow-up patients





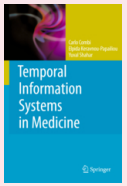
Dealing with temporalities of data in OO data models and query languages

- Direct use of the object oriented data model
- Modeling of the temporal dimensions of data through ad-hoc data models



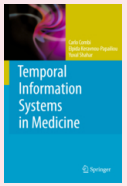
Direct use of the object oriented data model

- OODAPLEX and TIGUKAT adopt the direct use of an object-oriented data model.
- suitable data types (i.e., classes) allow the database designer to model temporal information.



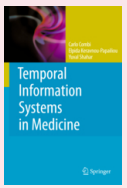
Direct use of the object oriented data model

- For example, in modeling different concepts of time, OODAPLEX allows the usage of the supertype *point*, from which each user-defined or system-supported data type inherits, to represent different temporal dimensions.
- TIGUKAT models the valid time at the level of object and of collections of objects.
- However, for each single application it is possible to use the rich set of system-supported classes, to define the real semantics of valid (or transaction) time.



Direct use of the object oriented data model

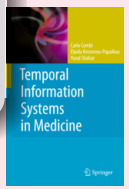
- The query language defined in TIGUKAT is able to follow the more recent standards existing for SQL; the proposed query language TQL extends SQL statements to support the object-oriented data model of TIGUKAT
- in a similar way to that proposed in OODAPLEX, TQL doesn't define specific constructs to explicitly manage temporal aspects of a query.
- Among the proposed object-oriented query languages, OQL, proposed by the ODMG, gained some interest both in the research and the commercial areas: it has a syntax similar to that of SQL, even though it is based on a fully fledged object data model.



Example

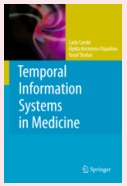
The physician wants to determine the names and the enrollment dates of those patients who had heart rate values higher than 100 bpm during visits occurred at least 10 days after the enrollment of patients.
The query can be expressed as follows, according to the OQL syntax:

```
SELECT P.surname, P.enroll_Date
FROM Patient P, P.Follow-up V
WHERE V.hR > 100 AND
      V.vis_date > (P.enroll_Date + 10 days)
```

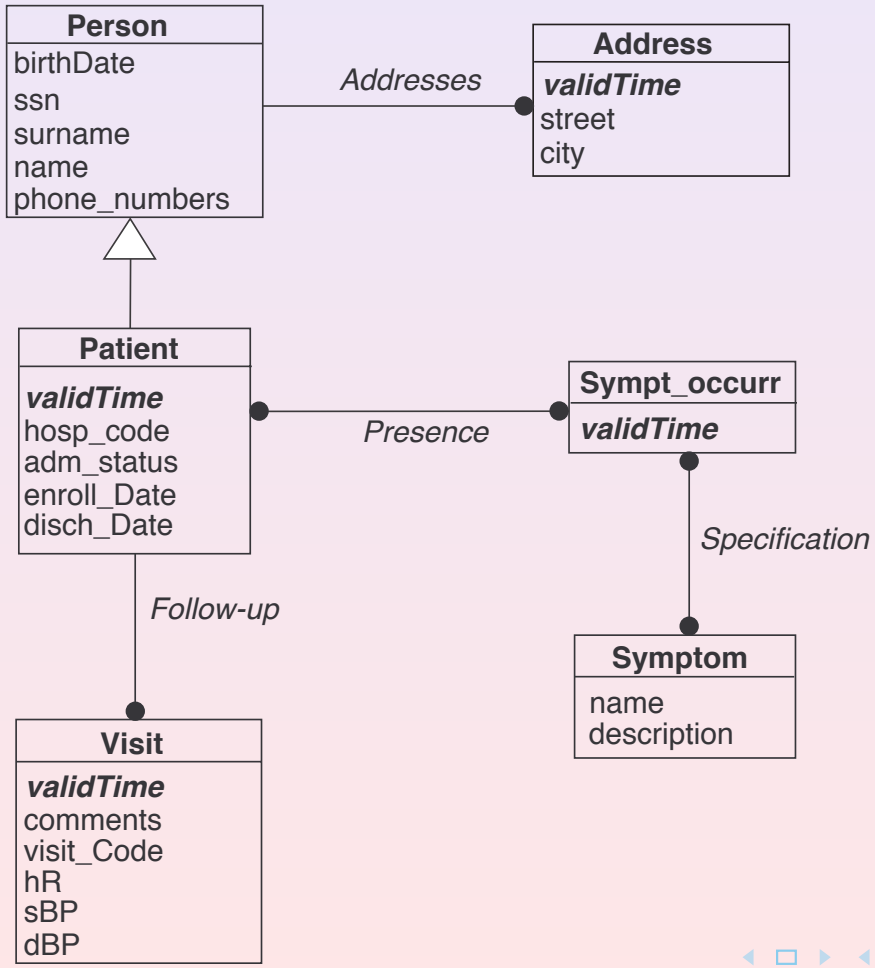


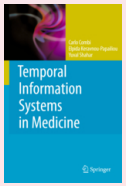
Temporally extended OO data models

Specifying temporally-oriented extensions of object oriented data models and query languages, to allow the user to explicitly model and consider temporal dimensions of data.



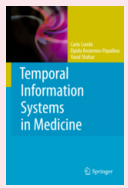
Temporal OO schema of the database about follow-up patients





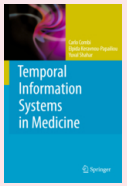
Temporally extended OO data models

- The temporal data model underlying this notation allows the distinction between *atemporal* classes (such as *Symptom*), which do not have a temporal dimension, and *temporal* classes (such as *Patient*), which have a valid time.
- The data model provides a sound semantics for temporal classes, i.e. classes having the attribute *validTime*, and the user can focus on the application-dependent aspects of the database schema.
- In other words, In general, both classes, associations, and attributes could be extended to manage temporally-oriented features of information.



Temporally extended OO data models: TOOSQL

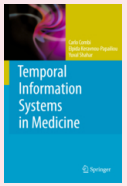
- TOOSQL supports both valid and transaction time.
- TOOSQL data model encodes attribute values as time sequences, composed of (*value*, *temporal element*) pairs.
- Proposed extensions to SQL are related to all the statements, both for query and for updating.



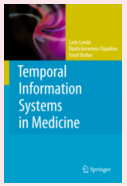
Temporally extended OO data models: TOOSQL

The `SELECT` statement has some new clauses

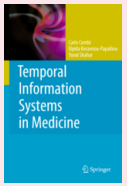
- `WHEN`, to define constraints on the valid time of retrieved objects;
- `GROUP TO`, to link new temporal sequences to other temporal sequences previously stored in the database;
- `MOVING WINDOW`, to consider aggregate data (e.g. the mean salary) with respect to an interval having a specified duration and moving on the valid time of an attribute;
- `TIME SLICE`, to consider only those objects valid in the specified interval;
- `ROLLBACK` and `WITHOUT CORRECTIONS`, to query the database about the transaction time.



Outline

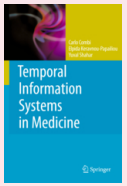


- In the medical area a huge amount of information comes in the form of images, videos, graphics, and even audio data, i.e., in the form of **multimedia data**.
- Another basic feature of medical and clinical data is that often they are stored according to different formats, with different structures, or even with missing or implicit structures: i.e., in the form of **semistructured data**, i.e., data that are neither raw data nor strictly typed data with an absolute schema fixed in advance.



Modeling multimedia data

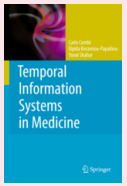
- integration of textual and visual information
- temporal aspects of textual and visual data and of their relationships
  - static images
  - video data
  - audio data
- temporal multimedia data models
  - temporal aspects in multimedia presentations



Modeling temporalities of video and images

In modeling video and images we can distinguish different levels of abstraction:

1. at the lowest abstraction level video data is stored as an uninterpreted (raw) stream of bytes.
2. At an higher level of abstraction a video stream is perceived as a flow of images (frames), displayed at a suitable frequency (frame-rate).
3. A further level of abstraction arises if we assume that in a video stream we can identify, according to its content, different subparts, which can be combined with other subparts of other video streams to obtain new videos.

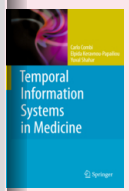


Modeling temporalities of video and images

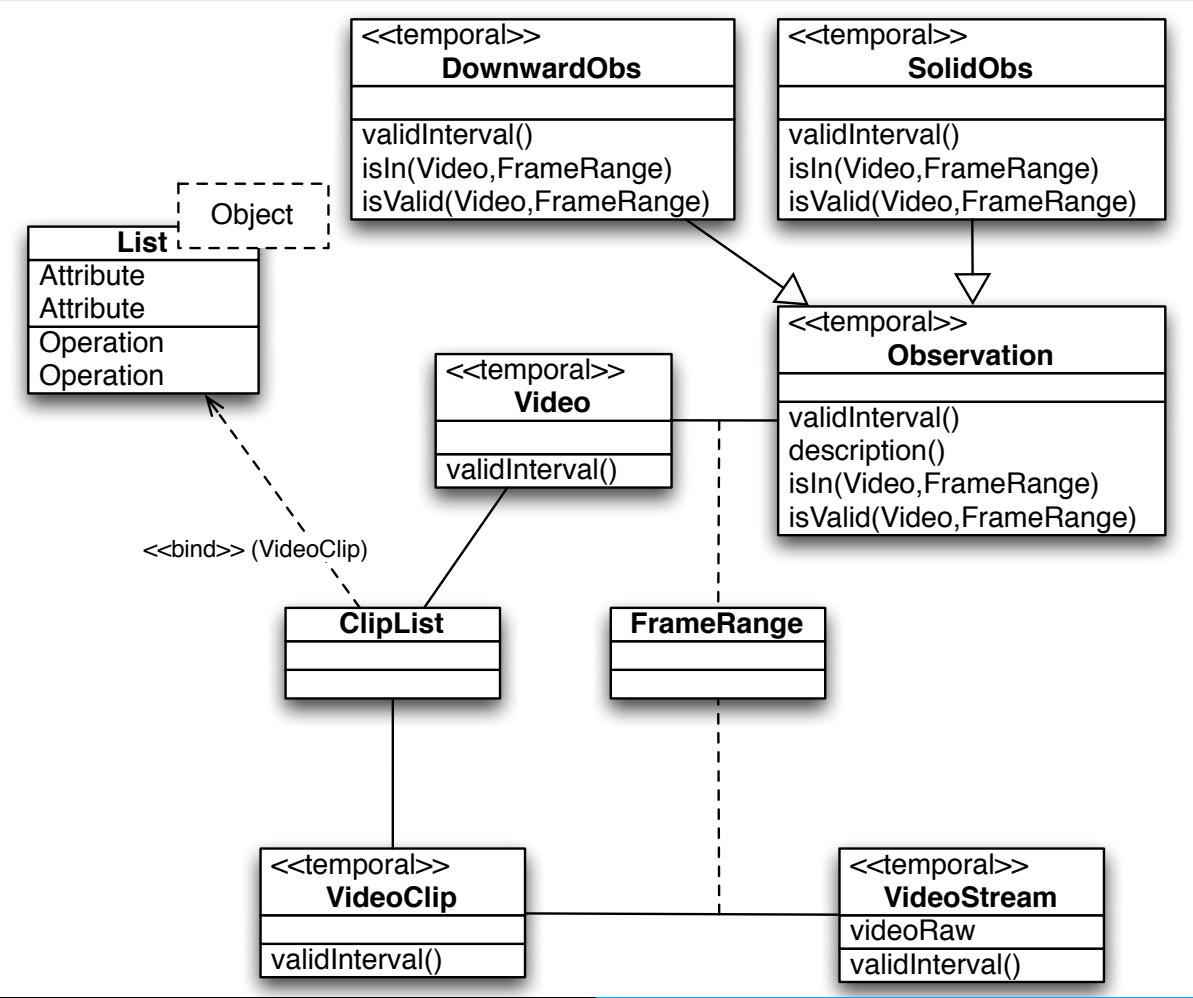
- Usually, an object is simply associated to (i.e., present in) a sequence of frames, and thus to each subsequence of the sequence.
- A more sophisticated approach has been proposed to be able to deal with different semantics of the associations of objects, or more generally observations, and frame subsequences.

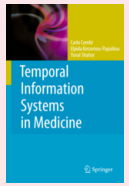
Example

The observation “cardiac cycle of the patient during a cardiac angiography” maintains its proper meaning only if associated to the whole identified frame sequence and not to some subsequence.



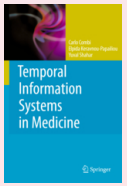
The UML schema for a temporal multimedia OO database





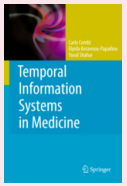
A temporal OO multimedia data model

- Classes `Video`, `VideoClip`, and `VideoStream` realize three different levels of abstraction.
- Valid times of objects of abstract classes are derived from the valid times of the composing low level raw objects, representing video streams.
- Classes `Observation` and their two subclasses `SolidObs` and `DownwardObs` represent different temporal semantics of observations related to videos.



A temporal OO multimedia data model

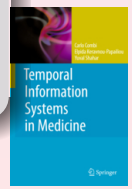
- Methods `isIn` and `isValid`, redefined in each class inheriting from `Observation`, allow one to specify whether an observation is simply associated to a frame range or it even holds on it.
- The class `FrameRange` represents the specification of both the association between videos and observations and the association between videoclips and videostreams.



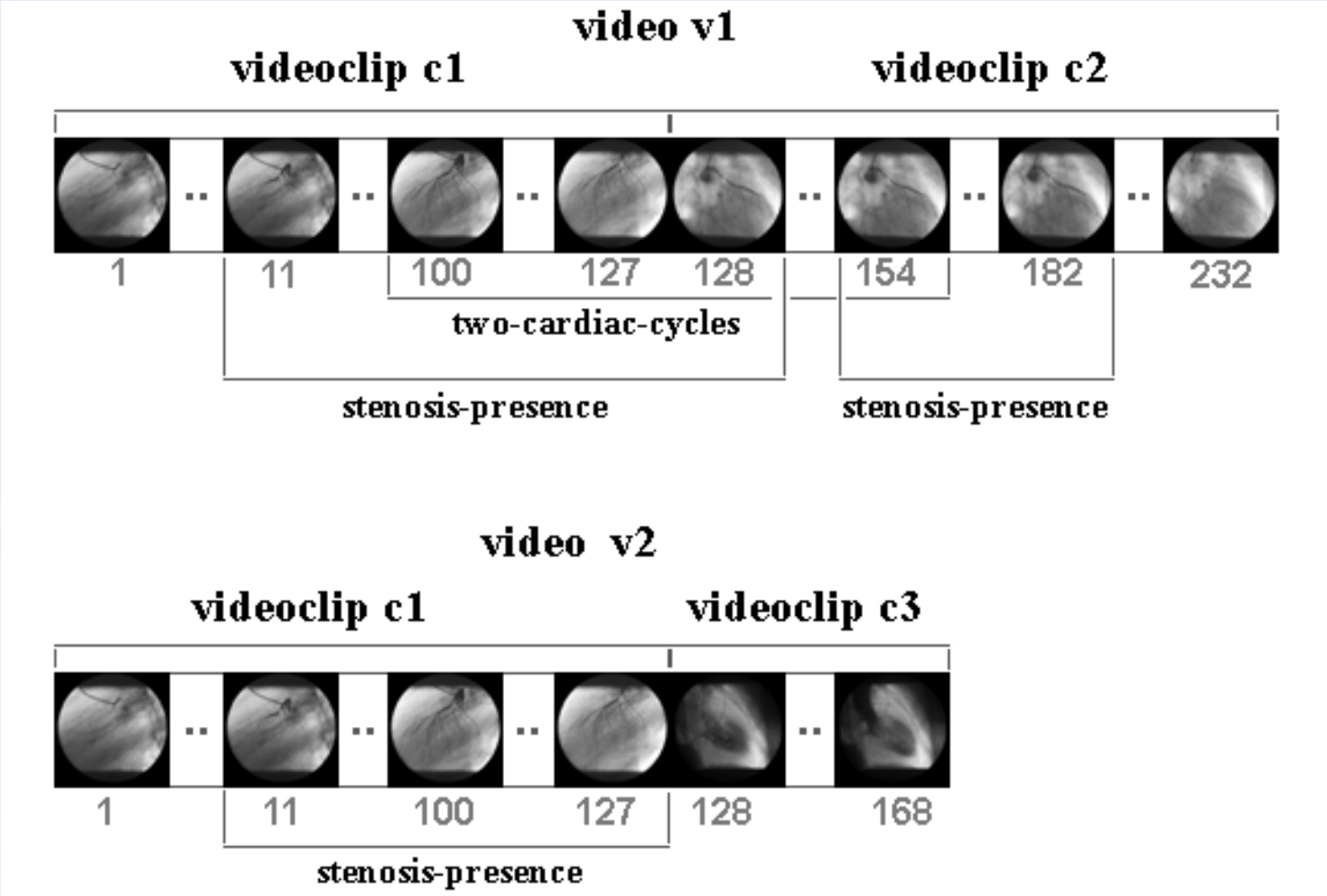
Modeling temporalities of angiocardiography exams

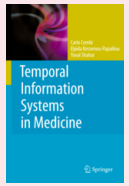
Example

- Cardiac angiography is a technique adopted to study the situation of coronary vessels (coronary angiography) and of heart functionalities (left ventriculography);
- The result of a cardiac angiography consists of an X-ray movie;
- Diagnoses based on the content of the movie consist of identifying stenoses (i.e., reductions of vessel lumen) and problems in the movement (contraction/relaxation) of the heart.



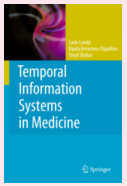
Modeling temporalities of angiocardiography exams





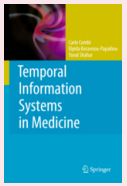
Temporalities and semistructured data

- As for the classical database field, also in the *semistructured data* context it has been considered the possibility to extend previously proposed models in order to take into account the evolution of data through time.
- Representing and querying changes in semistructured data and specifically in XML data have been often based on graph-based data models.

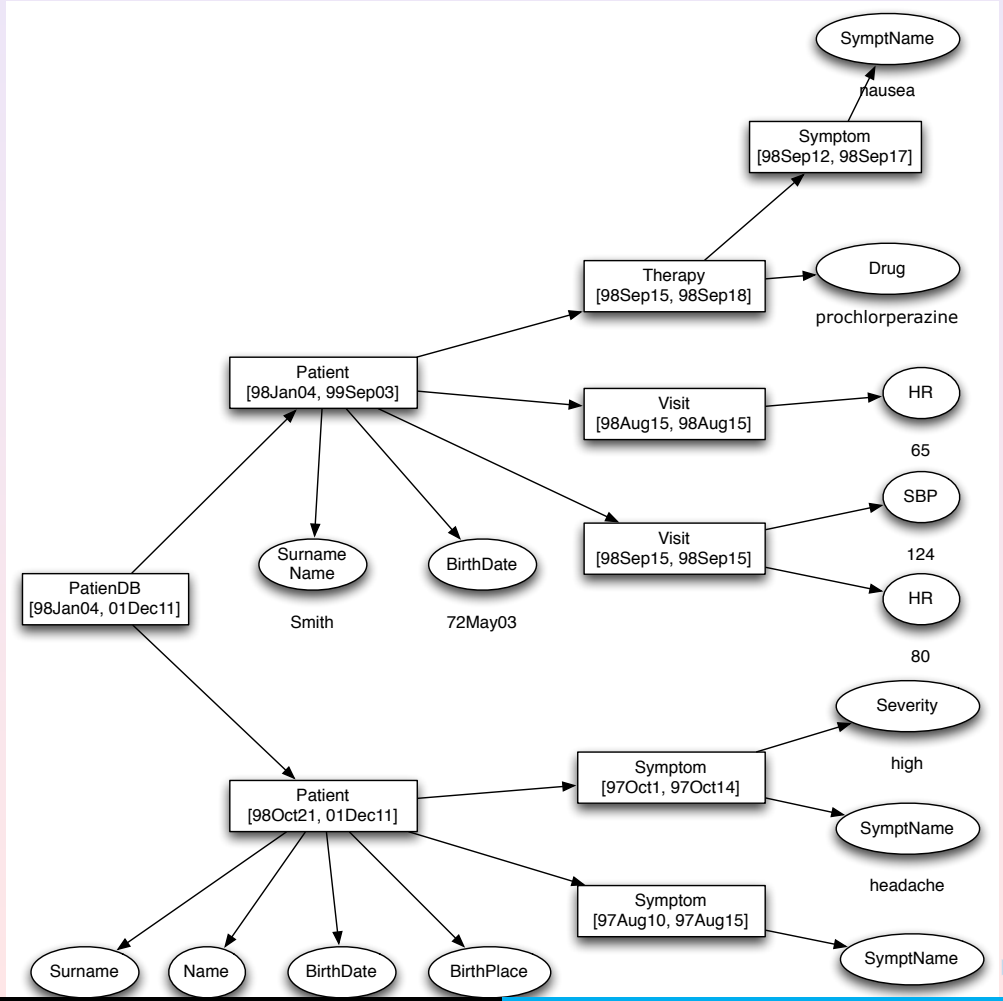


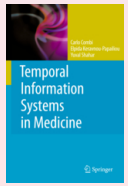
Temporalities and semistructured data

- Semistructured temporal data models can provide the suitable infrastructure for an effective management of time-varying documents on the web.
- Semistructured temporal data models could be considered as the reference model for the logical design of (data-intensive) web sites, when time-dependent information plays a key-role.



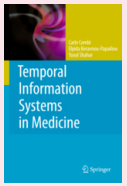
A temporal semistructured database for symptoms, therapies, and visits



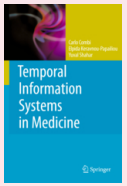


A temporal semistructured data model

- Only non-leaf nodes (represented through a box) have a temporal dimension, namely the valid time.
- Both schema and instance come together in the graph-based representation.
- Moreover, the tree-structure allows one to represent different co-existing possible structures.
- Different constraints should be considered among valid times of different nodes.

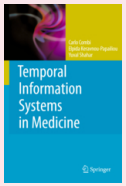


Outline



Take Home Summary

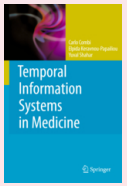
- Fundamental notions of temporal databases.
  - valid and transaction times
  - the taxonomy for temporal databases
- How to extend the relational model and the related algebra and calculus.
- Extending the widely known SQL.
- Extensions to the ER data model and object-oriented models and languages extended to consider temporal information.
- Temporal aspects of multimedia and semistructured data.



Main Bibliographic Reference

Carlo Combi, Elpida Keravnou-Papailiou, and Yuval Shahar.
Temporal Information Systems in Medicine.
Springer, 2010.

Several further references inside the book

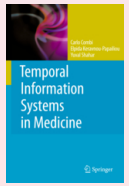


References on temporal dimensions of data

Carlo Combi and Angelo Montanari.
Data models with multiple temporal dimensions: Completing the picture.
In K. R. Dittrich, A. Geppert, and M. C. Norrie, eds, *CAiSE*, volume 2068 of *LNCS*, pages 187–202. Springer, 2001.

Sharma Chakravarthy and Seung-Kyum Kim.
Resolution of time concepts in temporal databases.
Inf. Sci., 80(1-2):91–125, 1994.

Christian S. Jensen et Al.
The consensus glossary of temporal database concepts - february 1998 version.
In *Temporal Databases, Dagstuhl*, pages 367–405, 1998.

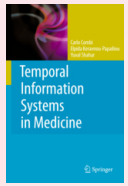


References on temporal data models

Carlo Combi, Sara Degani, and Christian S. Jensen.
Capturing Temporal Constraints in Temporal ER Models.
In: ER 2008. LNCS, vol. 5231, pages 397–411. Springer, 2008.

Christian S. Jensen and Richard T. Snodgrass.
Temporal specialization and generalization.
IEEE Trans. Knowl. Data Eng., 6(6):954–974, 1994.

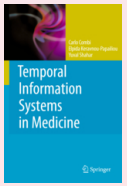
Gultekin Özsoyoglu and Richard T. Snodgrass.
Temporal and real-time databases: A survey.
IEEE Trans. Knowl. Data Eng., 7(4):513–532, 1995.



References on temporal extensions to SQL

Richard T. Snodgrass.
Developing Time-Oriented Database Applications in SQL.
Morgan Kaufmann Publishers, 2000.

Cindy Xinmin Chen, Jiejun Kong, and Carlo Zaniolo.
Design and implementation of a temporal extension of SQL.
In U. Dayal, K. Ramamritham, and T. M. Vijayaraman, eds, *ICDE*, pages 689–691. IEEE Computer Society, 2003.



References on temporal extensions of SQL

Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen, and Andreas Steiner.
Transitioning temporal support in TSQL2 to SQL3.
In *Temporal Databases, Dagstuhl*, pages 150–194, 1997.

Jose Ramon Rios Viqueira and Nikos A. Lorentzos.
SQL extension for spatio-temporal data.
VLDB J., 16(2):179–200, 2007.

Esteban Zimányi.
Temporal aggregates and temporal universal quantification in standard SQL.
SIGMOD Record, 35(2):16–21, 2006.