

Part3

Representational Temporal Data Models, Algebras and Query Languages

- ▶ Representational Temporal Data Models
- ▶ The Timestamp Now
- ▶ Representational Temporal Algebra
- ▶ Concrete Temporal Query Languages

Literature and Acknowledgements

- ▶ M. H. Böhlen, C. S. Jensen, Temporal Data Model and Query Language Concepts, Encyclopedia of Information Systems, Volume 4, Elsevier Science, 2003.
- ▶ M. H. Böhlen, J. Gamper, C. S. Jensen, Temporal Databases, unpublished draft, 2009.
- ▶ C. S. Jensen, M. D. Soo, and R. T. Snodgrass, Unifying temporal data models via a conceptual model. *Information Systems*, vol. 19, no. 7, pp. 513-547, 1994.
- ▶ These slides are based on slides from
 - ▶ Johann Gamper, Free University of Bozen-Bolzano, Italy
 - ▶ Christian S. Jensen, Aalborg University, Denmark

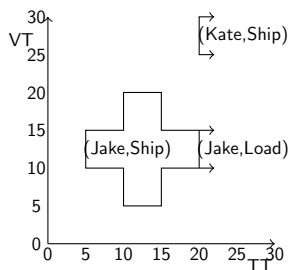
Terminology

- ▶ The terms *abstract* and *conceptual* have been used for issues that are independent of the implementation.
- ▶ The terms *concrete* and *representational* have been used for issues that consider the implementation.
- ▶ We use the respective terms interchangeably and follow the most commonly used practice.
- ▶ In general it is good practice to separate semantics (abstract, conceptual) and implementation (concrete, representational).

Representational Temporal Data Models

- ▶ The BCDM versus representational models
- ▶ Snodgrass' tuple timestamping model
- ▶ Jensen's backlog model
- ▶ Ben-Zvi's tuple timestamped model
- ▶ Gadia's attribute value timestamped model

The BCDM



dept (at TT = 19)

Emp	Dept	T
Jake	Ship	$\{(5,10), \dots, (5,15), \dots, (9,10), \dots, (9,15), (10,5), \dots, (10,20), \dots, (14,5), \dots, (14,20), (15,10), \dots, (15,15), \dots, (19,10), \dots, (19,15)\}$
Jake	Load	$\{(now,10), \dots, (now,15)\}$
Kate	Ship	$\{(now,25), \dots, (now,30)\}$

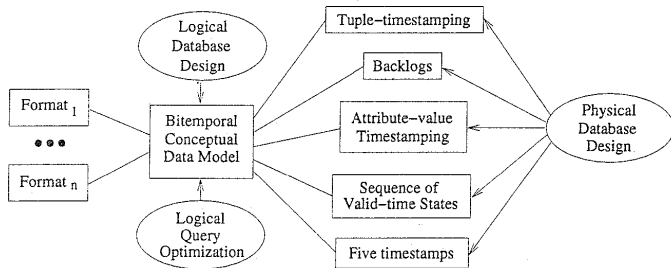
- Note 1: the BCDM defines the semantics of all statements; semantics not foreseen by the BCDM are not allowed
- Note 2: the BCDM only supports point semantics; it cannot be used for interval semantics

The BCDM versus Representational Data Models

- ▶ A BCDM relation is conceptually and structurally simple
 - ▶ a set of facts, each timestamped with a bitemporal element
- ▶ Aimed for the design and as a basis for query languages, rather than for implementation
- ▶ Representational data models are designed with implementation in mind

Display Formats

Representational Data Models



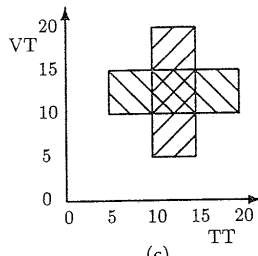
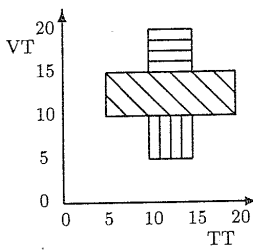
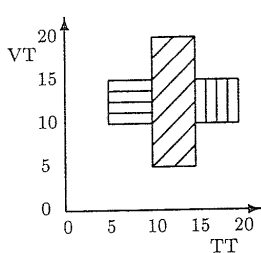
Snodgrass' Tuple Timestamped Data Model/1

▶ Snodgrass' tuple timestamped data model

- ▶ Supports valid time and transaction time
- ▶ Adds four **atomic-valued attributes** to each relation
 - ▶ Start and end point of the valid time (V_s, V_e)
 - ▶ Start and end point of the transaction time (T_s, T_e)
- ▶ Schema: $R = (A_1, \dots, A_n, T_s, T_e, V_s, V_e)$
- ▶ T_s, T_e, V_s, V_e represent the bitemporal chronons of the corresponding **rectangular** region
- ▶ 1NF relations

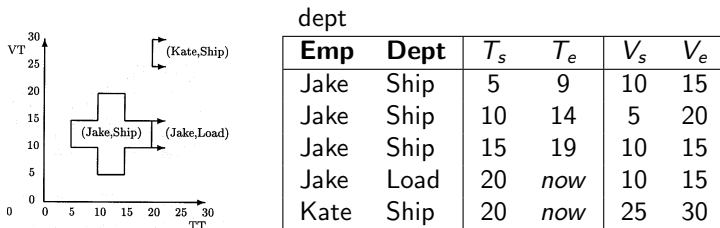
Snodgrass' Tuple Timestamped Data Model/2

- ▶ **Representation** of a BCDM tuple x in Snodgrass' representational model
 1. Divide the region covered by the BCDM timestamp into rectangles
 2. Represent a BCDM tuple by a set of representational tuples, one for each rectangle
- ▶ **Covering function:** Divides a bitemporal region $x[T]$ into rectangles s.t.
 - ▶ any bitemporal chronon in $x[T]$ is contained in at least one rectangle
 - ▶ each bitemporal chronon in a rectangle is contained in $x[T]$
- ▶ Various coverings of a bitemporal element are possible
 - ▶ Overlapping versus non-overlapping rectangles
 - ▶ Partitioning by transaction time versus partitioning by valid time



Snodgrass' Tuple Timestamped Data Model/3

- **Example:** Department relation in the Snodgrass data model, using partitioning by transaction time.



- Partitioning by transaction time yields maximal segments in valid time direction
- Partitioning by valid time yields maximal segments in transaction time direction

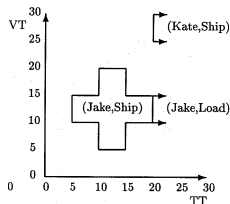
Jensen's Backlog-Based Data Model/1

► Jensen's backlog-based data model

- Supports valid time and transaction time
- Adds four **atomic-valued attributes** to each relation
 - Start and end point of the valid time (V_s, V_e)
 - Transaction time when the tuple was inserted into the backlog (T)
 - An operation which is either insert or delete (Op)
- Schema: $R = (A_1, \dots, A_n, V_s, V_e, T, Op)$
- Tuples in backlogs are never updated, i.e., backlogs are **append-only**
- 1NF relations
- The fact in an insertion request is current starting at the transaction's timestamp and until a matching delete request is recorded.

Jensen's Backlog-Based Data Model/2

- **Example:** Department relation in Jensen's backlog model



dept

Emp	Dept	V_s	V_e	T	Op
Jake	Ship	10	15	5	I
Jake	Ship	10	15	10	D
Jake	Ship	5	20	10	I
Jake	Ship	5	20	15	D
Jake	Ship	10	15	15	I
Jake	Ship	10	15	20	D
Jake	Ship	10	15	20	I
Kate	Ship	25	30	20	I

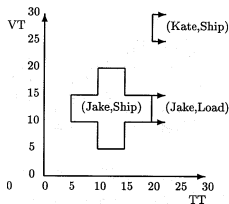
Ben-Zvi's Tuple Timestamped Data Model/1

► Ben-Zvi's Tuple Timestamped Data Model

- Adds five **atomic-valued attributes** to each relation
 - T_{es} is the *effective start* of the non-temporal attributes values
 - T_{ee} is the *effective end* of the non-temporal attributes values
 - T_{rs} is the *registration start* and stores when the T_{es} value was recorded
 - T_{re} is the *registration end* and stores when the T_{ee} value was recorded
 - T_{ee} is not necessarily recorded at the same time as T_{es} is recorded
 - T_d indicates the time when the information in the tuple was logically deleted
- The symbol “—” indicates
 - an unrecorded $T_{ee}(T_{re})$ value
 - in the T_d field that the tuple contains current information
- Schema: $R = (A_1, \dots, A_n, T_{es}, T_{rs}, T_{ee}, T_{re}, T_d)$
- 1NF relations

Ben-Zvi's Tuple Timestamped Data Model/2

- **Example:** Department relation in Ben-Zvi's tuple timestamped data model



dept

Emp	Dept	T_{es}	T_{rs}	T_{ee}	T_{re}	T_d
Jake	Ship	10	5	15	5	10
Jake	Ship	5	10	20	10	15
Jake	Ship	10	15	15	15	20
Jake	Load	10	20	15	20	—
Kate	Ship	25	20	30	20	—

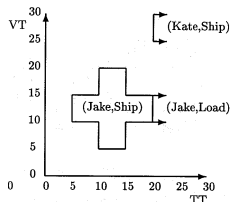
Gadia's Attribute Value Timestamped Data Model/1

► Gadia's attribute value timestamped data model

- Supports valid time and transaction time
- Schema: $R = (\{([T_s, T_e] \times [V_s, V_e] A_1)\}, \dots, \{([T_s, T_e] \times [V_s, V_e] A_n)\})$
- A tuple is composed of n sets
 - Each set element is a triple of a transaction-time period, a valid-time period, and an attribute value
- Non-1NF relations
- A relation might be restructured (grouped) on different attributes
 - for example, group by department rather than employee yields facts for each department

Gadia's Attribute Value Timestamped Data Model/2

- **Example:** Department relation in Gadia's attribute value timestamped data model



dept

Emp		Dept	
$[5,9] \times [10,15]$	Jake	$[5,9] \times [10,15]$	Ship
$[10,14] \times [5,20]$	Jake	$[10,14] \times [5,20]$	Ship
$[15,19] \times [10,15]$	Jake	$[15,19] \times [10,15]$	Ship
$[20,\text{now}] \times [10,15]$	Jake	$[20,\text{now}] \times [10,15]$	Load
$[20,\text{now}] \times [25,30]$	Kate	$[20,\text{now}] \times [25,30]$	Ship

Summary of Temporal Data Models

Data model	Citation	Time	Remark
–	[Brooks 56]	valid	The first to consider time in databases
Time Oriented Data Base Model	[Wiederhold et al. 75]	valid	Wiederhold was the first temporal model to be implemented; mainly to support medial applications.
DATA	[Kimball 78]	transaction	First implemented transaction-time model
LEGOL 2.0	[Jones & al. 79]	valid	Jones was the first to define a time-oriented algebra.
Time Relational Model	[Ben-Zvi 82]	both	First to incorporate both transaction time and valid time.
Historical Data Model	[Clifford & Warren 83]	valid	Attempted to model the semantics of natural language; associate time with with both attribute values and tuples.
–	[Snodgrass & Ahn 86]	both	
Temporally Oriented Data Model	[Ariav 86]	valid	
HQuel	[Tansel 86]	valid	Operators to switch between period and point representation
Historical Relational Data Model	[Clifford & Croker 87]	valid	
TQuel	[Snodgrass 87]	both	
Temporal Data Model	[Segev & Shoshani 87]	valid	
HQL	[Sadeghi 87]	valid	
Postgres	[Stonebraker 87]	transaction	
Homogeneous Relational Model	[Gadia 88]	valid	
Heterogeneous Relational Model	[Gadia & Yeung 88]	valid	
Temporal Relational Model	[Lorentzos 88]	valid	Operators to switch between period and point representation; supports nested granularity timestamps and periodic time.
Temporal Relational Model	[Navathe & Ahmed 89]	valid	
HSQL	[Sarda 90]	valid	Operators to switch between period and point representation
DM/T	[Jensen et al. 91]	transaction	
–	[McKenzie & Snodgrass 91]	both	
Accounting Data Model	[Thompson 91]	both	
TempSQL	[Gaida 92]	both	

Now in Representational Temporal Data Models

- ▶ Now in the BCDM
- ▶ Interpretation of Now in Representational Models

Now in Representational Data Models/1

- ▶ The BCDM allows *now* only for transaction time.
- ▶ The BCDM is not directly applicable if *now* is used as a valid timestamp.
- ▶ It is relatively straightforward to extend the BCDM to allow *now* in valid time timestamps.
- ▶ Instead of *now* the timestamp UC (until changed) has been used for transaction time.
- ▶ It states that the fact keeps being true until it is changed by a delete or update operation.
- ▶ We use *now* for transaction and valid time.

Now in Representational Data Models/2

The BCDM relation at time 19 (remember that the BCDM relation changes as time passes by).

dept (at TT = 19)

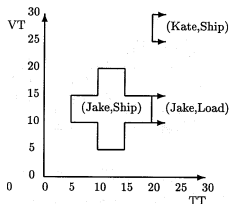
Emp	Dept	T
Jake	Ship	{(5,10),..., (5,15),..., (9,10),..., (9,15), (10,5),..., (10,20),..., (14,5),..., (14,20), (15,10),..., (15,15),..., (19,10),..., (19,15) }
Jake	Load	{(now,10),..., (now,15)}
Kate	Ship	{(now,25),..., (now,30)}

dept (at TT = 20)

Emp	Dept	T
Jake	Ship	{(5,10),..., (5,15),..., (9,10),..., (9,15), (10,5),..., (10,20),..., (14,5),..., (14,20), (15,10),..., (15,15),..., (19,10),..., (19,15) }
Jake	Load	{(20,10),..., (20,15), (now,10),..., (now,15)}
Kate	Ship	{(20,25),..., (20,30), (now,25),..., (now,30)}

Now in Representational Data Models/3

The concrete relation instance (we assume Snodgrass tuple timestamped model) does not change as time passes by.

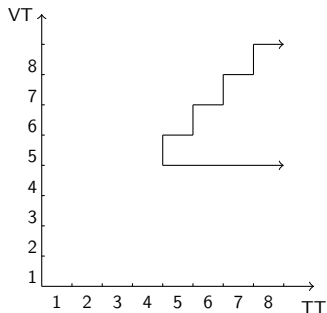


dept

Emp	Dept	T_s	T_e	V_s	V_e
Jake	Ship	5	9	10	15
Jake	Ship	10	14	5	20
Jake	Ship	15	19	10	15
Jake	Load	20	<i>now</i>	10	15
Kate	Ship	20	<i>now</i>	25	30

The Timestamp NOW/1

E-ID	EName	Salary	Dept	VT	TT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	3-now	5-now
1	Jane	30	Advtg	1-4	1-now
2	John	20	Sales	1-now	1-4

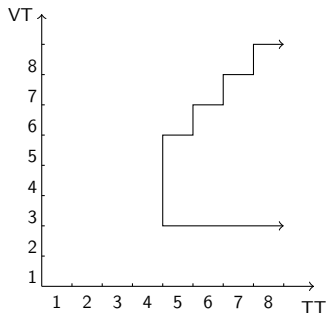


A growing stair shape:

- ▶ Starting at time 5, the valid time extends from 5 to now.
- ▶ At each clock tick, the period end grows by one unit.

The Timestamp NOW/2

E-ID	EName	Salary	Dept	VT	TT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	3-now	5-now
1	Jane	30	Advtg	1-4	1-now
2	John	20	Sales	1-now	1-4

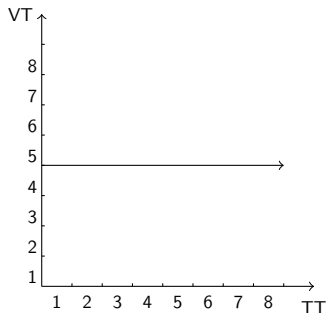


A growing stair shape with an initial big step:

- ▶ Starting at time 5, the valid time extends from 3 to now.
- ▶ At each clock tick, the period end grows by one unit.

The Timestamp NOW/3

E-ID	EName	Salary	Dept	VT	TT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	3-now	5-now
1	Jane	30	Advtg	1-4	1-now
2	John	20	Sales	1-now	1-4

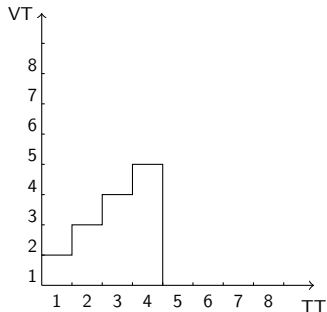


A growing period:

- ▶ Starting at time 1, the valid time extends from 1 to 4.
- ▶ The period remains part of the current state.
- ▶ The right end of the region tracks the current time.

The Timestamp NOW/4

E-ID	EName	Salary	Dept	VT	TT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	3-now	5-now
1	Jane	30	Advtg	1-4	1-now
2	John	20	Sales	1-now	1-4



A static stair shape:

- ▶ Starting at time 1, the valid time extends from 1 to now until time 4.
- ▶ The shape grew until time 4, but is now static.

Implications of Timestamp NOW

- ▶ When *now* is stored as a timestamp then the database state changes continuously as time passes by.
- ▶ Some facts stop being valid as time passes by.
- ▶ Some facts start being valid as time passes by.
- ▶ Index structures must also be dynamic (cf. TPR-Tree)
- ▶ The database integrity might get violated as time passes by.
- ▶ Materialized views (precomputed results) get outdated as time passes by.
- ▶ The timestamp *now* cannot be approximated by NULL or infinity (e.g., a query that retrieves all periods that are between 5 and 7 years long).

Representational Temporal Algebra

- ▶ Algebra for Snodgrass' tuple timestamped data model
- ▶ Semantic equivalence and snapshot equivalence
- ▶ Reducibility of operators

Algebra for Snodgrass' Tuple Timestamped Model/1

- ▶ Relation schema: $R = (A_1, \dots, A_n, T_s, T_e, V_s, V_e)$
 - ▶ V is a shorthand for (V_s, V_e) , T is a shorthand for (T_s, T_e)
- ▶ **Temporal projection:** Project a relation r with non-timestamp attributes $A_1, \dots, A_n$ to a subset D of these attributes (columns)

$$\pi_D^B(r) = \{z^{(|D|+4)} \mid \exists x \in r(z[D] = x[D] \wedge z[T] = x[T] \wedge z[V] = x[V])\}$$

- ▶ Note:
 - ▶ The result of a projection might not be coalesced (even if the argument relation is coalesced).
 - ▶ $\{(C101, T1234, 7, now, 8, 10), (C101, T1245, 7, now, 11, 15)\}$

Algebra for Snodgrass' Tuple Timestamped Model/2

- ▶ **Temporal selection:** Select from relation r with non-timestamp attributes $A_1, \dots, A_n$ all tuples that satisfy predicate P defined on the non-timestamp attributes

$$\pi_P^B(r) = \{z^{(n+4)} \mid z \in r \wedge P(z[A])\}$$

- ▶ Note:
 - ▶ Predicates on timestamps are important and would be reasonable.
 - ▶ $P = (\text{duration}(V, \text{year}) > 3)$
- ▶ **Temporal union:** Compute the union of tuples from two relations r_1 and r_2 that are instances of the same schema (or union-compatible schemas)

$$r_1 \cup^B r_2 = \{z^{(n+4)} \mid \exists x \in r_1 \exists y \in r_2 (z = x \vee z = y)\}$$

- ▶ Note:
 - ▶ The result of a union might not be coalesced (even if the argument relations are coalesced).

Algebra for Snodgrass' Tuple Timestamped Model/3

- **Temporal difference:** Computes those tuples that are in r_1 and not in r_2 . The two relations are instances of the same schema (or union-compatible schemas).

$$\begin{aligned} r_1 -^B r_2 = \{ & z^{(n+4)} \mid \exists x \in r_1 (z[A] = x[A]) \wedge \\ & \exists t \in \text{cover}(\text{bi_chr}(x[T], x[V]) - \\ & \quad \{ \text{bi_chr}(y[T], y[V]) \mid y \in r_2 \wedge y[A] = x[A] \}) \wedge \\ & z[T_s] = \text{min_1}(t) \wedge z[T_e] = \text{max_1}(t) \wedge \\ & z[V_s] = \text{min_2}(t) \wedge z[V_e] = \text{max_2}(t)) \} \end{aligned}$$

- The new timestamp is determined by set difference of bitemporal elements
- $\text{bi_chr}(T, V)$ computes the bitemporal chronons covered by the argument rectangular region
- $\text{cover}(X)$ produces a set of rectangles that together cover all the bitemporal chronons of X .

Algebra for Snodgrass' Tuple Timestamped Model/4

- ▶ **Temporal join:** Two tuples join if they match on the join attributes and have overlapping bitemporal-element timestamps.
 - ▶ r and s are instances over the following schemas:

$$\begin{aligned}R &= (A_1, \dots, A_n, B_1, \dots, B_l, T_s, T_e, V_s, V_e) \\S &= (A_1, \dots, A_n, C_1, \dots, C_m, T_s, T_e, V_s, V_e)\end{aligned}$$

$$\begin{aligned}r \bowtie^B s = \{ & z^{(n+l+m+4)} \mid (\exists x \in r \exists y \in s (z[A] = x[A] = y[A] \wedge \\ & x[T] \cap y[T] \neq \emptyset \wedge x[V] \cap y[V] \neq \emptyset \wedge \\ & z[B] = x[B] \wedge z[C] = y[C] \wedge \\ & z[T] = x[T] \cap y[T] \wedge z[V] = x[V] \cap y[V])) \}\end{aligned}$$

► Timeslide operators

$$\rho_{t_1}^B(r) = \{z^{(n+2)} \mid \exists x \in r(z[A] = x[A] \wedge z[V] = x[V] \wedge t_1 \in x[T])\}$$

$$\tau_{t_2}^B(r) = \{z^{(n+2)} \mid \exists x \in r(z[A] = x[A] \wedge z[T] = x[T] \wedge t_2 \in x[V])\}$$

- $\rho_{t_1}^B$ is the transaction timeslice operator. It selects the valid time relation at transaction time t_1 .
- $\tau_{t_2}^B$ is the valid timeslice operator. It selects the transaction time relation at valid time t_2 .
- $\rho_{t_1}^T$ is the transaction timeslice of a transaction time relation.
- $\tau_{t_2}^V$ is the valid timeslice of a valid time relation.

Semantic Equivalence/1

- ▶ Many semantically equivalent representations of the same conceptual relation may exist.
- ▶ We need a mechanism to determine if two instances
 - ▶ are represented differently but have the same meaning
 - ▶ are represented differently and indeed have different meaning
- ▶ **Snapshot equivalence:** Two bitemporal relations, r and s , are snapshot equivalent, $r \stackrel{s}{\equiv} s$, if for all times t_1 (not exceeding the current time) and for all times t_2 the following holds:

$$\tau_{t_2}^V(\rho_{t_1}^B(r)) = \tau_{t_2}^V(\rho_{t_1}^B(s))$$

Semantic Equivalence/2

- ▶ **Theorem:** Let r and s be BCDM relations over the same schema. Then $r \stackrel{s}{\equiv} s$ iff $r = s$.
- ▶ **Theorem:** Let r be a temporal relation. Then for all times t_1 not exceeding the current time and for all times t_2 we can swap the timeslice operators, i.e.,

$$\tau_{t_2}^V(\rho_{t_1}^B(r)) = \rho_{t_1}^T(\tau_{t_2}^B(r))$$

- ▶ In **point-based** and **weak period-based** data models:
 - ▶ Snapshot equivalent relations have the **same information content** (i.e., are semantically equivalent).
- ▶ Snapshot equivalence is **not applicable in strong period-based models!**
 - ▶ Snapshot equivalent relations do not have the same information content.

Semantic Equivalence/3

- **Example:** Two-day and four-day checkouts in the video example

checkout1		
CustID	TapeNum	T
C102	T1245	[19,20]
C102	T1245	[21,22]

checkout2		
CustID	TapeNum	T
C102	T1245	[19,22]

- Apply the valid-timeslice operator at each time point

$$\tau_{19}^V(\text{checkout1}) = \{(C102, T1245)\}$$

$$\tau_{20}^V(\text{checkout1}) = \{(C102, T1245)\}$$

$$\tau_{21}^V(\text{checkout1}) = \{(C102, T1245)\}$$

$$\tau_{22}^V(\text{checkout1}) = \{(C102, T1245)\}$$

$$\tau_{19}^V(\text{checkout2}) = \{(C102, T1245)\}$$

$$\tau_{20}^V(\text{checkout2}) = \{(C102, T1245)\}$$

$$\tau_{21}^V(\text{checkout2}) = \{(C102, T1245)\}$$

$$\tau_{22}^V(\text{checkout2}) = \{(C102, T1245)\}$$

- checkout1 and checkout2 are snapshot equivalent
- checkout1 and checkout2 contain the **same information** under the *weak period-based semantics*, but **different information** under the *strong period-based semantics*
 - Snapshot equivalence adopts a point-based view

Reducibility of Temporal Operators/1

- ▶ Operators in the various data models (snapshot, transaction-time, valid-time, and bitemporal) are related.
 - ▶ The semantics of an operator in a more complex data model can be **reduced** to the semantics of the operator in a simpler data model.
- ▶ Reducibility guarantees that the semantics of simpler operators are preserved in their more complex counterparts.
- ▶ **Theorem:** Let t denote an arbitrary time and let r and s be relation instances of the proper types for the operators they are applied to. Then the following holds for all t .

$$\begin{aligned}\rho_t^T(r \bowtie^T s) &\stackrel{s}{=} \rho_t^T(r) \bowtie^S \rho_t^T(s) \\ \tau_t^V(r \bowtie^V s) &\stackrel{s}{=} \tau_t^V(r) \bowtie^S \tau_t^V(s) \\ \tau_t^B(r \bowtie^B s) &\stackrel{s}{=} \tau_t^B(r) \bowtie^T \tau_t^B(s) \\ \rho_t^B(r \bowtie^B s) &\stackrel{s}{=} \rho_t^B(r) \bowtie^T \rho_t^B(s)\end{aligned}$$

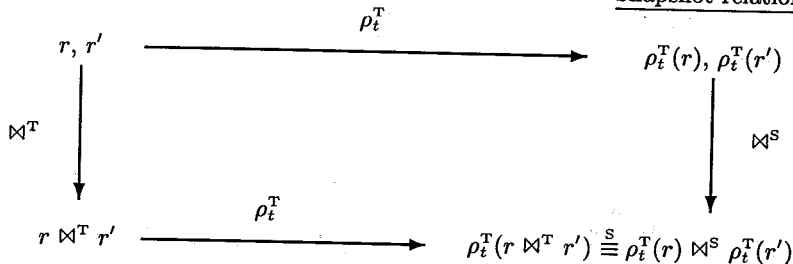
- ▶ Similar theorems for the other operators.

Reducibility of Temporal Operators/2

- **Example:** Reducibility of transaction-time natural join to snapshot natural join
 - The result of first joining two transaction-time relations and then transforming the result to a snapshot relation yields a result that is the same as that obtained by first transforming the arguments to snapshot relations and then joining the snapshot relations.

Transaction-time relations

Snapshot relations



Concrete Temporal Query Languages

- ▶ Language design criteria
- ▶ Comparison of timestamps
- ▶ Comparative analysis
 - ▶ **SQL + abstract data types**: extend SQL with abstract data type
 - ▶ **IXSQL**: two operators to convert periods to points and vice versa
 - ▶ **SQL/TP**: only points exist; periods are used internally and for display.
 - ▶ **TSQL2**: use syntactic defaults to easily formulate temporal statements
 - ▶ **ATSQL**: use semantic defaults to easily formulate temporal statements

Temporal Query Languages

- ▶ A **temporal query language**, QL , is a part of a temporal data model:
 $DM = (DS, QL)$
 - ▶ Operates on the data structure DS
 - ▶ Define/create temporal relations
 - ▶ Insert/delete/modify information in a temporal relation
- ▶ Many temporal extensions of SQL have been proposed
- ▶ The goal is not to learn these languages but to understand and appreciate the concepts represented by these languages.

Language Design Criteria/1

- ▶ **Expressive power**

- ▶ Suitable for intended applications

- ▶ **Consistency**

- ▶ Upward compatibility with standards, e.g., SQL standard
- ▶ Internally consistent with various aspects of the language itself
- ▶ Consistent syntax (e.g., not mix up postfix and prefix notation)

- ▶ **Clarity**

- ▶ Syntax should reflect the semantics → facilitates the formulation and understanding of queries
- ▶ Consistent naming style
- ▶ Zero-One-Infinity principle (the only reasonable numbers in a programming language design are zero, one, and infinity)

Language Design Criteria/2

- ▶ **Minimality**

- ▶ As few as possible new reserved words and clauses
- ▶ No duplication of functionality

- ▶ **Orthogonality**

- ▶ Possibility to freely combine query language constructs that are semantically independent
- ▶ Zero-One-Infinity principle

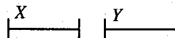
- ▶ **Closed-form evaluation**

- ▶ The result of the query can be represented in the language of the database

Comparison of Timestamps/1

- ▶ Convenient **comparison of timestamps** is essential for a temporal query language
- ▶ Many query languages adopt (a subset of) **Allen's 13 period relations**

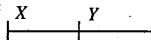
X before Y
Y after X



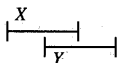
X equals Y



X meets Y
Y met_by X



X overlaps Y
Y overlapped_by X



X during Y
Y contains X



X starts Y
Y started_by X



X finishes Y
Y finished_by X



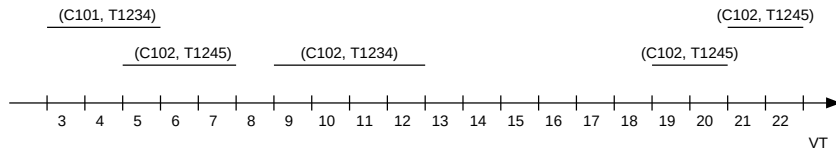
Comparison of Timestamps/2

► Overview of **period comparison operators** in TSQL, HSQL, and TQuel

Allen	TSQL	HSQL	TQuel
a before b	a BEFORE b or b AFTER a	a PRECEDES b	a precede b and not (end of a equal begin of b)
a equals b	a EQUIVALENT b	a = b	a equal b
a meets b	a PRECEDES b or b FOLLOWS a	a MEETS b	end of a equal begin of b
a overlaps b	a OVERLAP b AND a.TIME-END PRECEDES b.TIME-END	a OVERLAP b AND a.TO < b.TO	a overlap b and end of a precedes end of b
a during b	a DURING b	b CONTAINS a	begin of a overlap b and end of a overlap b and not (a equal b)
a meets b or b meets a	a ADJACENT b	a ADJACENT b	end of a equal begin of b or end of b equal begin of a
a starts b	(a.TIME-START EQUIVALENT b.TIME-START) AND (a.TIME-END PRECEDES b.TIME-END)	a.FROM = b.FROM AND a.TO < b.TO	begin of a equal begin of b and end of a precede end of b
a finishes b	(a.TIME-START AFTER b.TIME-START) AND (a.TIME-END EQUIVALENT b.TIME-END)	a.FROM > b.FROM AND a.TO = b.TO	begin of a precede begin of b and end of a equal end of b

Video Rental Example/1

- **Example:** Video store where customers (CustID) rent video tapes (TapeNum). The rentals are stored in a valid-time relation Checkout.



Video Rental Example/2



- Different queries on the Checkout table with their intended result:

Q1: *All check-outs that overlap the time period [7, 9].*

CustID	TapeNum	T
C102	T1245	[5,7]
C102	T1234	[9,12]

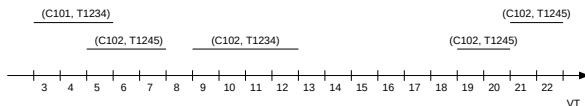
Q2: *All 2-day check-outs.*

TapeNum	T
T1245	[19,20]
T1245	[21,22]

Q3: *Which tapes were checked out together with tape T1234?*

TapeNum	T
T1245	[5,5]

Video Rental Example/3



Q4: What is the number of tapes that have been checked out?

Cnt	T
1	[3,4]
2	[5,5]
1	[6,7]
0	[8,8]
1	[9,12]
0	[13,18]
:	:
:	:

Q5: How many check-outs were made in total?

Cnt
5

Q6: List all (current) check-outs (assume current time = 5).

CustID	TapeNum
C101	T1234
C102	T1245

SQL + Abstract Data Types/1

- ▶ Extend existing language (e.g., SQL) with **time data types** and associated predicates and functions
 - ▶ e.g., predicates for timestamp comparison
- ▶ Earliest and (from a language design perspective) simplest approach
- ▶ Has **limited impact on existing language** and well understood formalism
- ▶ Assume **period-timestamped** relations

SQL + Abstract Data Types/2

Q1: `select *`
`from CheckOut`
`where T overlaps [7,9]`

Q2: `select TapeNum, T`
`from CheckOut`
`where duration(T) = 2`

Q3: `select b.TapeNum, intersect(a.T, b.T) as T`
`from CheckOut as a, CheckOut as b`
`where a.TapeNum = T1234 and`
`b.TapeNum <> T1234 and`
`a.T overlaps b.T`

Q5: `select count(*) from CheckOut`

Q6: `select CustID, TapeNum`
`from CheckOut`
`where T overlaps [now]`

- Query Q4 is very complicated in SQL

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

Assessment of SQL + Abstract Data Types

- ▶ An abstract data type for periods is useful.
- ▶ The gain from having a period instead of start and end point is not big.
- ▶ An abstract data type does not offer a **systematic way to generalize** snapshot queries to temporal queries
 - ▶ Queries Q3 and Q4 both generalize a simple snapshot query to becoming time-varying.
 - ▶ The only difference is in the snapshot query, a join (Q3) versus an aggregation (Q4)
 - ▶ However, techniques used to formulate Q3 may not be re-used when formulating Q4 and vice versa.
- ▶ New solutions must be invented for new kinds of queries

- ▶ IXSQL extends SQL2 with **(time) period data**
- ▶ Periods are convenient for representing temporal aspects, but create difficulties in formulating temporal queries
- ▶ Avoid these problems by **normalizing timestamps** so that they are **aligned** (identical or disjoint)
 - ▶ Function **unfold**: decompose a period-timestamped tuple into a set of point-timestamped tuples (one for each point in the original period)
 - ▶ Function **fold**: collapse a set of point timestamped tuples into value-equivalent tuples timestamped with maximum periods
- ▶ **General pattern** for query processing using fold/unfold:
 1. Construct the point-based representation by unfolding the argument relation(s)
 2. Compute the query on point-based representation
 3. Fold the result to end up with an period-based representation

IXSQL/2

- ▶ The formulation of the first 3 queries are essentially SQL
- ▶ Unfold/fold mechanism is not required

Q1: `select *
from CheckOut
where T cp [7,9]`

Q2: `select TapeNum, T
from CheckOut
where dur (T) = 2`

Q3: `select b.TapeNum, intervsect(a.T, b.T) as T
from CheckOut as a, CheckOut as b
where a.TapeNum = T1234 and
b.TapeNum <> T1234 and
a.T cp b.T`

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

- Unfold/fold mechanism becomes useful for temporal aggregation (Q4)

```
Q4:  select count(*), T
      from    ( select *
                  from CheckOut
                  reformat as unfold T )
      group by T
      reformat as fold T
```

```
Q5:  select count(*)
      from    CheckOut
```

```
Q6:  select CustID, TapeNum
      from    CheckOut
      where   T overlaps [now]
```

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

Assessment of IXSQL

- ▶ Only two functions, fold and unfold, are added to SQL. This is attractive.
- ▶ **Efficient evaluation** of queries formulated using fold/unfold has yet to be resolved
 - ▶ Unfolding has a worst case exponential space complexity
- ▶ Neither purely **point-based** nor **period-based** view
 - ▶ Sensitive to specific period representation of data (e.g., queries that do not use fold/unfold)
 - ▶ Different results for different but snapshot-equivalent relations
 - ▶ Fold/unfold only preserve information of a point-based view
 - ▶ Normalization step using unfold/fold loses period information
 - ▶ Fold is not the inverse of unfold (information about the original periods is lost)

SQL/TP .../1

- ▶ A **point-based** temporal extension of SQL
- ▶ Defined on temporal relations that use point timestamps
- ▶ Simple, unambiguous, and **well-defined semantics**

SQL/TP .../2

- ▶ Q1 needs a comparison of neighbor DB states
 - ▶ Point-based view separates database states
 - ▶ First approach: `select CustID, TapeNum from CheckOut where T=7 or T=8 or T=9`
 - ▶ Fails! Restricts the time of the original checkout period to 7, 8, 9.
 - ▶ Join over SeqNo is required!

Q1: `select a.*
from CheckOut a, CheckOut b
where a.SeqNo = b.SeqNo
 (b.T = 7 or b.T = 8 or b.T = 9)`

- ▶ Q2 can only be answered if SeqNo is present
 - ▶ Formulate duration query as aggregate query

Q2: `select SeqNo, TapeNum, T
from CheckOut
group by SeqNo
having count(T) = 2`

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	3
1	C101	T1234	4
1	C101	T1234	5
2	C102	T1245	5
2	C102	T1245	6
2	C102	T1245	7
3	C102	T1234	9
3	C102	T1234	10
3	C102	T1234	11
3	C102	T1234	12
4	C102	T1245	19
4	C102	T1245	20
5	C102	T1245	21
5	C102	T1245	22

SQL/TP .../3

- ▶ Strength of SQL/TP is to extend snapshot queries to temporal queries if only a single state is involved
 - ▶ Extend snapshot query with equality constraint on timestamp

Q3: `select b.TapeNum, b.T
from CheckOut as a, CheckOut as b
where a.TapeNum = T1234 and
b.TapeNum <> T1234 and
a.T = b.T`

Q4: `select count(*), T
from CheckOut
group by T`

- ▶ An additional attribute is needed for time-invariant aggregation
 - ▶ `select count(*) from CheckOut` fails for Q5!

Q5: `select count(distinct SeqNo) from CheckOut`

Q6: `select CustID, TapeNum from CheckOut where T = now`

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	3
1	C101	T1234	4
1	C101	T1234	5
2	C102	T1245	5
2	C102	T1245	6
2	C102	T1245	7
3	C102	T1234	9
3	C102	T1234	10
3	C102	T1234	11
3	C102	T1234	12
4	C102	T1245	19
4	C102	T1245	20
5	C102	T1245	21
5	C102	T1245	22

Assessment of SQL/TP.../1

- ▶ **Additional attribute needed** to express some queries and preserve information, e.g., periods of the argument relations
- ▶ Additional attribute is **not a systematic approach** to obtain point-based semantics **and** preservation of periods
- ▶ Example: *When was tape T1245, but not tape T1234, checked out?*

- ▶ Expected answer: {[6, 7], [19, 20], [21, 22]}

- ▶ Plausible solution:

```
select T
from   CheckOut
where  TapeNum = T1245
except
select T from Checkout
where  TapeNum = T1234
```

- ▶ Does not guarantee that (fragments of) individual check-out periods are returned because the sequence numbers are not included
- ▶ Including the sequence number disables the difference

Assessment of SQL/TP.../2

- ▶ **Period representation is still needed** for user interaction and internal presentation
 - ▶ What the user sees is periods (points are coalesced for presentations to the user)
 - ▶ For formulating queries the user needs to (conceptually) map periods to points.
- ▶ Some timestamps, e.g., $(C101, T1234, [22, \infty])$ lead to problems (infinite relations).

- ▶ Extension of SQL2 to support **time periods** (weak semantics) and **time points**
- ▶ Adopts **syntactic defaults** that make the formulation of common temporal queries more convenient
- ▶ A syntactic default is an abbreviation for a more complex expression. Instead of the abbreviation it is always possible to use the complex expression instead.
- ▶ A **default valid clause** is implicitly placed after the select clause
 - ▶ Computes the intersection of the valid times of the argument relation(s) in the from clause
 - ▶ Intersection is returned as timestamp in the result relation

TSQL2/2

```
Q1:  select *
      from    CheckOut
      where   valid(T) overlaps period '7-9'
```

- ▶ Result in Q2 must include SeqNo, otherwise value-equivalent tuples are coalesced
- ▶ Cast function maps between periods and timestamps

```
Q2:  select SeqNo, TapeNum, T
      from    CheckOut
      where   cast(valid(T) as period) = 2
```

- ▶ Implicit default valid clause in Q3
 - ▶ valid intersect(valid(a),valid(b))
 - ▶ i.e., non intersecting tuples do not contribute to the result

```
Q3:  select b.TapeNum
      from    CheckOut as a, CheckOut as b
      where   a.TapeNum = T1234 and b.TapeNum <> T1234
```

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	[3,5]
2	C102	T1245	[5,7]
3	C102	T1234	[9,12]
4	C102	T1245	[19,20]
5	C102	T1245	[21,22]

TSQL2/3

Q4: `select COUNT(*)
from CheckOut`

- ▶ The snapshot keyword is used to retrieve non-temporal relations (the default result is a temporal relation)

Q5: `select snapshot COUNT(*)
from CheckOut`

Q6: `select snapshot *
from CheckOut
where valid(CheckOut) overlaps [now]`

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	[3,5]
2	C102	T1245	[5,7]
3	C102	T1234	[9,12]
4	C102	T1245	[19,20]
5	C102	T1245	[21,22]

Assessment of TSQL2

- ▶ TSQL2 is a **large language** with many parts and an **informally defined semantics**
- ▶ It is unclear what the syntactic defaults for Q4 should be (no formal definition)
- ▶ Syntactic defaults are **not scalable**
 - ▶ A default is required for each language construct
 - ▶ Difficult to be comprehensive and systematic and to ensure that defaults do not interact in unanticipated ways
- ▶ The language is **complex** and therefore difficult to understand and use

- ▶ ATSQL introduces **temporal statement modifiers** to add temporal support to SQL
- ▶ Statement modifiers are **semantic defaults** that indicate the intended semantics without specifying how to compute it
- ▶ Aims to provide a **systematic way** of constructing temporal queries from nontemporal queries
 1. Formulate first the corresponding nontemporal query
 2. Then apply a statement modifier
- ▶ Example: Temporal join
 - ▶ Formulate the nontemporal join
 - ▶ Modifier ensures that the argument timestamps overlap and that the result timestamp is the intersection of the argument periods
- ▶ ATSQL assumes period-timestamped tuples together with a **strong period semantics**
 - ▶ Value-equivalent tuples with adjacent or overlapping periods are permitted

ATSQL/1

- ▶ The `seq vt` (“sequenced valid time”) modifier ensures that timestamps are returned as if the enclosed statement was evaluated on each state of the database. In addition interval information is preserved.
- ▶ The rest of the query is standard SQL

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

```
Q1:  seq vt
      select *
      from  CheckOut
      where T overlaps [7,9]
```

```
Q2:  seq vt
      select TapeNum
      from  CheckOut
      where duration(T) = 2
```

Q3: seq vt
select b.TapeNum
from CheckOut as a, CheckOut as b
where a.TapeNum = T1234 and
b.TapeNum <> T1234

Q4: seq vt
select count(*) from CheckOut

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

ATSQL/4

- ▶ The nseq vt (“nonsequenced valid time”) modifier indicates that what follows should be treated as regular SQL
- ▶ The rest of the query is standard SQL

```
Q5:  nseq vt
      select count(*)
      from   CheckOut
```

- ▶ A query without a modifier considers only the current state of the argument relations (i.e., NOW)
- ▶ Ensures that legacy queries on nontemporal relations are unaffected if the nontemporal relations are made temporal

```
Q6:  select *
      from   CheckOut
```

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

Assessment of ATSQL

- ▶ Language mechanism is **independent of the syntactic complexity** of the queries
- ▶ Represents a much **more fundamental change** to the language than other approaches, e.g., abstract data types.
- ▶ The semantics of SQL extended with statement modifiers has been defined.
- ▶ **Implementation** of modifiers has still to be solved (for example, for modifiers applied to stored procedures)

Summary

- ▶ Representational temporal data models
- ▶ The timestamp now
- ▶ Representational temporal algebra
- ▶ Concrete temporal query languages