

Temporal and Spatial Data Management

Fall 2019

Temporal Data Models

SL04

- ▶ Modeling temporal data
- ▶ Temporal data models
- ▶ Temporal query languages

Table of Contents

- ▶ **Modeling temporal data**
 - ▶ **Dimensions of time**
 - ▶ **Timestamping data**
- ▶ Temporal data models
 - ▶ Snodgrass' tuple timestamping model
 - ▶ Jensen's backlog model
 - ▶ Ben-Zvi's tuple timestamped model
 - ▶ Gadia's attribute value timestamped model
- ▶ Comparative analysis of temporal query languages
 - ▶ SQL + abstract data types
 - ▶ IXSQL
 - ▶ SQL/TP
 - ▶ TSQL2
 - ▶ ATSQL

Temporal Data Models/1

- ▶ **Data model:** M = (DS , QL)
 - ▶ DS is a set of data structures
 - ▶ QL is a language for querying and updating the data structures
- ▶ Example: relational data model is composed of relations and SQL
- ▶ Many extensions of the relational data model to support time have been proposed
- ▶ Several modeling aspects have to be considered
 - ▶ Different time dimensions
 - ▶ Different timestamp types
 - ▶ Attribute versus tuple timestamping

Temporal Data Models/2

The different modeling aspects lead to **subtle** and **difficult** issues. There are pros and cons in all cases. No single choice seems best and after more than 20 years no consensus has been reached!

- ▶ Which time dimensions do we want?
 - ▶ Transaction time, valid time, decision time, ...
- ▶ Which timestamps shall we use?
 - ▶ points, periods, sets of points, ...
- ▶ Which information shall be timestamped?
 - ▶ objects, facts, part of objects, ...

Dimensions of Time/1

- ▶ Time is **multi-dimensional**
 - ▶ Valid time
 - ▶ Transaction time
 - ▶ Publication time
 - ▶ Efficacy time
 - ▶ Decision time
 - ▶ etc.
- ▶ The key question: Which time aspects are **sufficiently important** so that they should be supported by the database system?
- ▶ There is a broad consensus that transaction time and valid time are the most important time dimensions.
- ▶ Transaction time is about to be incorporated into commercial database systems (law requires accountability, etc).

Dimensions of Time/2

SQL standard: Business time

- ▶ **Valid time:** The time a fact was/is/will be true in the modeled reality/mini-world
 - ▶ A fact is a statement that is either true or false
 - ▶ A relation is a collection of facts
 - ▶ Example: John has been hired on October 1, 2004

Dimensions of Time/2

- ▶ **Valid time:** The time a fact was/is/will be true in the modeled reality/mini-world
 - ▶ A fact is a statement that is either true or false
 - ▶ A relation is a collection of facts
 - ▶ Example: John has been hired on October 1, 2004
 - ▶ Valid time captures the time-varying states of the mini-world

Dimensions of Time/2

- ▶ **Valid time:** The time a fact was/is/will be true in the modeled reality/mini-world
 - ▶ A fact is a statement that is either true or false
 - ▶ A relation is a collection of facts
 - ▶ Example: John has been hired on October 1, 2004
 - ▶ Valid time captures the time-varying states of the mini-world
 - ▶ All facts have a valid time by definition, however, it might not be recorded in the database

Dimensions of Time/2

- ▶ **Valid time:** The time a fact was/is/will be true in the modeled reality/mini-world
 - ▶ A fact is a statement that is either true or false
 - ▶ A relation is a collection of facts
 - ▶ Example: John has been hired on October 1, 2004
 - ▶ Valid time captures the time-varying states of the mini-world
 - ▶ All facts have a valid time by definition, however, it might not be recorded in the database
 - ▶ Valid time is independent of the recording of the fact in a database

Dimensions of Time/2

- ▶ **Valid time:** The time a fact was/is/will be true in the modeled reality/mini-world
 - ▶ A fact is a statement that is either true or false
 - ▶ A relation is a collection of facts
 - ▶ Example: John has been hired on October 1, 2004
 - ▶ Valid time captures the time-varying states of the mini-world
 - ▶ All facts have a valid time by definition, however, it might not be recorded in the database
 - ▶ Valid time is independent of the recording of the fact in a database
 - ▶ Valid time is either bounded (does not extend until infinity) or unbounded (extends until infinity)

Dimensions of Time/3

SQL standard: system time

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
 - ▶ With transaction time deletions of facts are purely logical
 - ▶ the fact remains in the database, but ceases to be part of the database's current state.

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
 - ▶ With transaction time deletions of facts are purely logical
 - ▶ the fact remains in the database, but ceases to be part of the database's current state.
 - ▶ Transaction time captures the time-varying states of the database

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
 - ▶ With transaction time deletions of facts are purely logical
 - ▶ the fact remains in the database, but ceases to be part of the database's current state.
 - ▶ Transaction time captures the time-varying states of the database
 - ▶ Always bounded on both ends
 - ▶ Starts when the database is created (nothing was stored before)
 - ▶ Does not extend past now (no facts are known to have been stored in the future)

performance
is most
important
issue

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
 - ▶ With transaction time deletions of facts are purely logical
 - ▶ the fact remains in the database, but ceases to be part of the database's current state.
 - ▶ Transaction time captures the time-varying states of the database
 - ▶ Always bounded on both ends
 - ▶ Starts when the database is created (nothing was stored before)
 - ▶ Does not extend past now (no facts are known to have been stored in the future)
 - ▶ Basis for supporting accountability and “traceability” requirements, e.g., in financial, medical, legal applications.

Dimensions of Time/3

- ▶ **Transaction time:** The time when a fact is current/present in the database as stored data
 - ▶ Example: the fact “John was hired on October 1, 2004” was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
 - ▶ Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
 - ▶ With transaction time deletions of facts are purely logical
 - ▶ the fact remains in the database, but ceases to be part of the database's current state.
 - ▶ Transaction time captures the time-varying states of the database
 - ▶ Always bounded on both ends
 - ▶ Starts when the database is created (nothing was stored before)
 - ▶ Does not extend past now (no facts are known to have been stored in the future)
 - ▶ Basis for supporting accountability and “traceability” requirements, e.g., in financial, medical, legal applications.
 - ▶ Should be supplied automatically by the DBMS

Dimensions of Time/4

Task 4.1

- ▶ A data model can support none, one, two, or more of these time dimensions
 - ▶ **Snapshot** data model: None of the time dimensions is supported
 - ▶ Represents a single snapshot of the reality and the database
 - ▶ **Valid time** data model: Supports only valid time
 - ▶ **Transaction time** data model: Supports only transaction time
 - ▶ **Bitemporal** data model: Supports valid time and transaction time
- ▶ An important issue is what shall be the basic element of a temporal data model:
 - ▶ objects, relations, classes, tuples, etc
 - ▶ closure is important

result type?

how many timesteps?

value of timestep?

possible?

result type?

└ join of a vt and tt relation?

└ join of 2 vt time relations:

Timestamping/1

- ▶ A timestamp is a value that is **associated** with data in a database
 - ▶ Captures some temporal aspect, e.g., valid time, transaction time
 - ▶ Represented as one or more attributes/columns of a relation

Timestamping/1

- ▶ A timestamp is a value that is **associated** with data in a database
 - ▶ Captures some temporal aspect, e.g., valid time, transaction time
 - ▶ Represented as one or more attributes/columns of a relation
- ▶ Three different **types of timestamps** are widely used
 - ▶ Time points
 - ▶ Time periods
 - ▶ Temporal elements

Timestamping/1

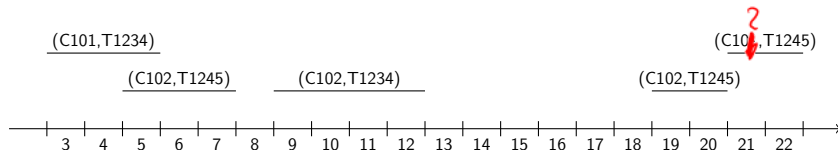
- ▶ A timestamp is a value that is **associated** with data in a database
 - ▶ Captures some temporal aspect, e.g., valid time, transaction time
 - ▶ Represented as one or more attributes/columns of a relation
- ▶ Three different **types of timestamps** are widely used
 - ▶ Time points
 - ▶ Time periods
 - ▶ Temporal elements
- ▶ Two different **ways of timestamping**
 - ▶ Tuple timestamping
 - ▶ Attribute timestamping

Timestamping/2

- ▶ **Example:** Video store where customers, identified by a `CustID`, rent video tapes, identified by a `TapeNum`. Consider the following rentals during May 1997:
 - ▶ On 3rd of May, customer `C101` rents tape `T1234` for three days
 - ▶ On 5th of May, customer `C102` rents tape `T1245` for 3 days
 - ▶ From 9th to 12th of May, customer `C102` rents tape `T1234`
 - ▶ From 19th to 20th of May, and again from 21st to 22nd of May, customer `C102` rents tape `T1245`

Timestamping/2

- ▶ **Example:** Video store where customers, identified by a `CustID`, rent video tapes, identified by a `TapeNum`. Consider the following rentals during May 1997:
 - ▶ On 3rd of May, customer C101 rents tape T1234 for three days
 - ▶ On 5th of May, customer C102 rents tape T1245 for 3 days
 - ▶ From 9th to 12th of May, customer C102 rents tape T1234
 - ▶ From 19th to 20th of May, and again from 21st to 22nd of May, customer C102 rents tape T1245
- ▶ These rentals are stored in a relation `CheckOut` which is graphically illustrated below



Tuple Timestamping with Points/1

exclude?
no. consecutive
CheckOut *rentals are lost*

- ▶ **Point-based data model:** each tuple is timestamped with a **time point/instant**.

- ▶ Most basic and simple data model.
- ▶ Timestamps are atomic values that can be easily compared, using $=, \neq, <, >, \geq, \leq$.
- ▶ Multiple tuples are used if a fact is valid at several time points.
- ▶ Syntactically different relations store different information.
- ▶ Provides an *abstract* view of a database and is not meant for physical implementation.
- ▶ Conceptual simplicity and computational complexity make it popular for theoretical studies.

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

Tuple Timestamping with Points/2

- ▶ The reconstruction of the original relation is not always possible.
- ▶ The table on the previous slide makes it impossible to determine if C102 rented T1245 once or twice in the period from 19 to 22.
- ▶ Additional attributes are required, e.g., SeqNo to represent the 2-day rentals.
- ▶ It is difficult to predict when an additional attribute is needed.

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

$\{(a), (b)\} - \{(a)\} = \{(b)\}$ no SeqID

$\{(1,a), (2,b)\} - \{(3,a)\} = \{(1,a), (2,b)\}$ with SeqID

Tuple Timestamping with Periods/1

- ▶ **Period-based data model** (usually the term interval-based is used): each tuple is timestamped with a **time period**.

CheckOut

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

- ▶ Timestamps are atomic values that can be compared using Allen's 13 basic relationships between periods (before, meets, during, etc.)
 - ▶ More convenient than comparing the endpoints of the periods.
 - ▶ The benefits of Allen's predicates are relatively small.

Tuple Timestamping with Periods/2

- ▶ The start and end of an interval are distinguished **change points**.
- ▶ The `SeqNo` attribute is not needed to distinguish different checkouts.
- ▶ Multiple tuples are used if a fact is valid over disjoint time periods
- ▶ Cannot model a single checkout with a gap.
- ▶ The most popular model from an implementation perspective.
- ▶ Time periods are not closed under all set operations.
 - ▶ Example: subtracting $[5, 7]$ from $[1, 9]$ returns a set of periods $\{[1, 4], [8, 9]\}$.

Tuple Timestamping with Temporal Elements

option?

*not all facts are stored
in one tuple*

- checkout facts ✓

- customer facts f

- tape facts f

- **Data models with temporal elements**: each tuple is timestamped with a **temporal element**, i.e., a finite union of periods

CheckOut

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

- The full history of a fact is stored in one tuple
- Usually the periods of a temporal element must be disjoint and non-adjacent. This makes it similar to point timestamps.

Attribute Timestamping/1

again?

- ▶ **Attribute value timestamping**: each **attribute value** is timestamped with a set of time points/periods
- ▶ Goal: capture all information about a real-world object in a single tuple
 - ▶ e.g., all information about a customer in the relation below

CheckOut

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

aggregate
vs
store all
aggregates

construct objects A
on the fly $T \rightarrow A$
by queries

we need QL support to get
flexibility; not different modeling

- Simplisity is lost (complex)

- real-world objects
 - customer ✓
 - tape &
 - checkout &

Attribute Timestamping/1

- ▶ **Attribute value timestamping:** each **attribute value** is timestamped with a set of time points/periods
- ▶ Goal: capture all information about a real-world object in a single tuple
 - ▶ e.g., all information about a customer in the relation below

CheckOut

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

- ▶ A single tuple may record multiple facts
 - ▶ e.g., the second tuple records the following facts: rental information for customer C102 for the tapes T1245 and T1234, and four different checkouts
- ▶ Non-first-normal-form data model

Attribute Timestamping/2

- ▶ **Different groupings** of the information into tuples are possible for attribute-value timestamping
 - ▶ Information about other objects is spread across several tuples, e.g., information about tapes.
 - ▶ e.g., grouping on tape number in the example below

CheckOut

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

⇒ regrouping operation

Table of Contents

- ▶ Modeling temporal data
 - ▶ Dimensions of time
 - ▶ Timestamping data
- ▶ **Temporal data models**
 - ▶ **Snodgrass' tuple timestamping model**
 - ▶ **Jensen's backlog model**
 - ▶ **Ben-Zvi's tuple timestamped model**
 - ▶ **Gadia's attribute value timestamped model**
- ▶ Comparative analysis
 - ▶ SQL + abstract data types
 - ▶ IXSQL
 - ▶ SQL/TP
 - ▶ TSQL2
 - ▶ ATSQL

Snodgrass' Tuple Timestamped Model/1

► Snodgrass' tuple timestamped data model

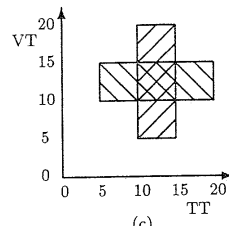
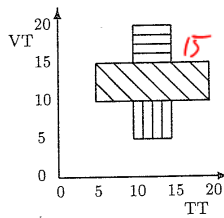
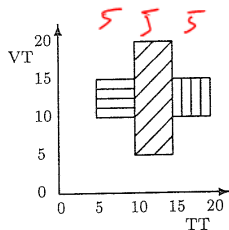
- Supports valid time and transaction time
- Adds four atomic-valued attributes to each relation
 - Start and end point of the valid time ($\underline{V_s}$, $\underline{V_e}$)
 - Start and end point of the transaction time ($\underline{T_s}$, $\underline{T_e}$)
- Schema: $R = (A_1, \dots, A_n, \underline{T_s},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

Snodgrass' Tuple Timestamped Model/2

- ▶ A closed region in a two dimensional space ($TT \times VT$) must be represented by a set of rectangles.
 - ▶ any bitemporal chronon in $x.T$ is contained in at least one rectangle
 - ▶ each bitemporal chronon in a rectangle is contained in $x.T$

Snodgrass' Tuple Timestamped Model/2

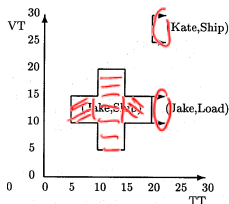
- ▶ A closed region in a two dimensional space ($TT \times VT$) must be represented by a set of rectangles.
 - ▶ any bitemporal chronon in $x.T$ is contained in at least one rectangle
 - ▶ each bitemporal chronon in a rectangle is contained in $x.T$
- ▶ Various coverings of a 2D area are possible:
 - ▶ Overlapping versus non-overlapping rectangles
 - ▶ Partitioning by transaction time versus partitioning by valid time



Q: how long was information in the DB before it was deleted?

Snodgrass' Tuple Timestamped Model/3

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

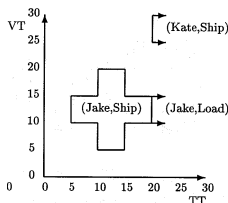


dept

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

Snodgrass' Tuple Timestamped Model/3

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



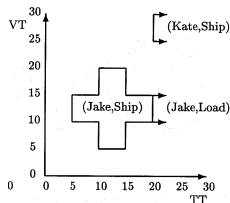
dept

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

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

The Timestamp NOW/1

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

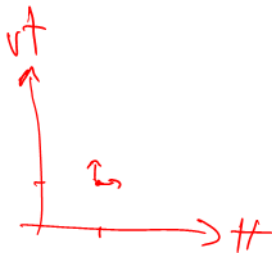


dept

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/2

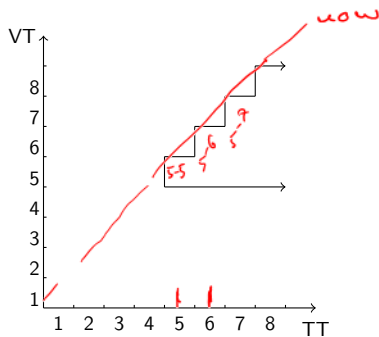
E-ID	EName	Salary	Dept	TT	VT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	5-now	3-now
1	Jane	30	Advtg	1-now	1-4
2	John	20	Sales	1-4	1-now



The Timestamp $NOW/2$ @TT=4 ~~5-4~~ @TT=5 5-5

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

@TT=6 5-6
@TT=7 5-7
⋮

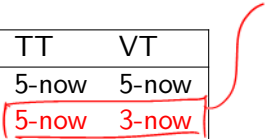


A growing stair shape:

- ▶ 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/3

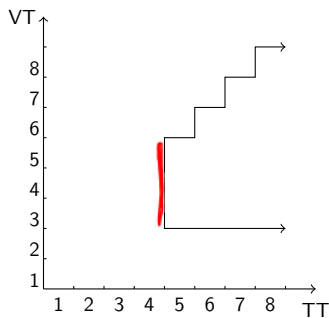
E-ID	EName	Salary	Dept	TT	VT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	5-now	3-now
1	Jane	30	Advtg	1-now	1-4
2	John	20	Sales	1-4	1-now



The Timestamp NOW/3

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

information was
recorded late
(after it became
valid in reality)



A growing stair shape with an initial big step:

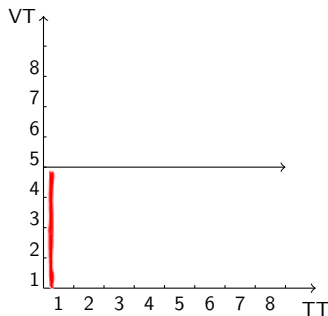
- ▶ Starting at time 5, the valid time extends from 3 to now.
- ▶ At each clock tick, the period end grows by one unit.

The Timestamp NOW/4

E-ID	ENAME	SALARY	DEPT	TT	VT
1	Jane	40	Advtg	5-now	5-now
2	John	20	Sales	5-now	3-now
1	Jane	30	Advtg	1-now	1-4
2	John	20	Sales	1-4	1-now

The Timestamp NOW/4

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



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.

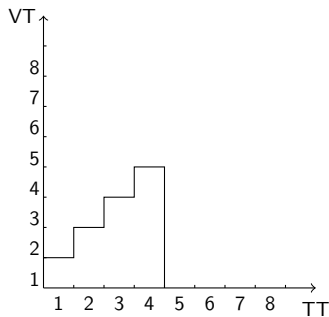
The Timestamp NOW/5

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



The Timestamp NOW/5

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



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.

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) *fine-grained*
- ▶ The database integrity might get violated as time passes by.
- ▶ Materialized views (precomputed results) get outdated as time passes by.

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).

Algebra for Snodgrass' Data Model/1

- cardinality (= # of attributes) = cardinality of D + 4 (start and end of VT and TT)
- 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)

Original

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

D values are the same

TT is the same

VT is the same

(of any tuple x and result tuple z)

Algebra for Snodgrass' Data 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).

$\pi_{\overline{8}}^8 \rightarrow \pi_{\text{CustID}}^8(r)$

▶ $\{(C101, T1234, 7, \text{now}, 8, 10), (C101, T1245, 7, \text{now}, 11, 15)\}$

C101	7-now	8-10
C101	7-now	11-15

- **Temporal natural join**: Two tuples join if they match on the join attributes and have overlapping timestamps.

- r and s are instances over the following schemas:

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

$$r \bowtie^B s = \{z^{(n+l+m+4)} \mid (\exists x \in r \exists y \in s (\underbrace{z.A = x.A = y.A}_{\text{A attrs are the same: } x, y, z} \wedge \underbrace{x.T \cap y.T \neq \emptyset \wedge x.V \cap y.V \neq \emptyset}_{\substack{\text{TT and VT of } x \text{ and } y \\ \text{must overlap}}} \wedge \underbrace{z.B = x.B \wedge z.C = y.C}_{\substack{\text{B values from } x \\ \text{C values from } y}} \wedge \underbrace{z.T = x.T \cap y.T \wedge z.V = x.V \cap y.V}_{\substack{\text{TT of } z \text{ is intersection of TT of } x \text{ and } y}})) \}$$

- Note:

- The result of a temporal bitemporal join is again a bitemporal relation.
- The original timestamps are not included in the result relation.

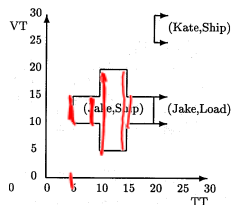
Jensen's Backlog-Based Data Model/1

▶ 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: $R = (A_1, \dots, A_n, \underline{T_{es}, T_{rs}, T_{ee}, T_{re}, T_d})$
- 1NF relations

vt start V_s

vt end V_e

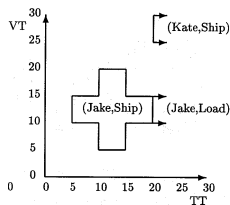
tt start T_s

tt end T_e

tt start T_s

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	—

could be different
($\Rightarrow$ more powerful)

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

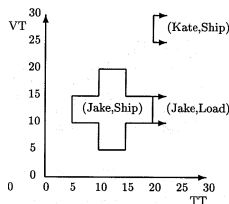
- A relation might be restructured (grouped) on different attributes

- for example, group by department rather than employee yields facts for each department

Gadia's Attribute Value Timestamped Data Model/2

Task 4.5

- **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

Table of Contents

- ▶ Modeling temporal data
 - ▶ Dimensions of time
 - ▶ Timestamping data
- ▶ Temporal data models
 - ▶ Snodgrass' tuple timestamping model
 - ▶ Jensen's backlog model
 - ▶ Ben-Zvi's tuple timestamped model
 - ▶ Gadia's attribute value timestamped model
- ▶ **Comparative analysis**
 - ▶ **SQL + abstract data types**
 - ▶ **IXSQL**
 - ▶ **SQL/TP**
 - ▶ **TSQL2**
 - ▶ **ATSQL**

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

- ▶ **Expressive power**
 - ▶ Suitable for intended applications
 - ▶ Economy of encoding is relevant

Language Design Criteria

- ▶ **Expressive power**
 - ▶ Suitable for intended applications
 - ▶ Economy of encoding is relevant
- ▶ **Clarity**
 - ▶ Syntax should reflect the semantics
 - ▶ Consistent naming style

Language Design Criteria

- ▶ **Expressive power**

- ▶ Suitable for intended applications
- ▶ Economy of encoding is relevant

- ▶ **Clarity**

- ▶ Syntax should reflect the semantics
- ▶ Consistent naming style

- ▶ **Consistency**

- ▶ Upward compatibility with standards, e.g., SQL standard
- ▶ Systematic (not a new construct per query query, no exceptions)

Language Design Criteria

▶ **Expressive power**

- ▶ Suitable for intended applications
- ▶ Economy of encoding is relevant

▶ **Clarity**

- ▶ Syntax should reflect the semantics
- ▶ Consistent naming style

▶ **Consistency**

- ▶ Upward compatibility with standards, e.g., SQL standard
- ▶ Systematic (not a new construct per query query, no exceptions)

▶ **Orthogonality**

- ▶ Possibility to freely combine query language constructs
- ▶ Zero-One-Infinity principle (the only reasonable numbers in a programming language design are zero, one, and infinity)

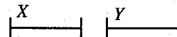
Language Design Criteria

- ▶ **Expressive power**
 - ▶ Suitable for intended applications
 - ▶ Economy of encoding is relevant
- ▶ **Clarity**
 - ▶ Syntax should reflect the semantics
 - ▶ Consistent naming style
- ▶ **Consistency**
 - ▶ Upward compatibility with standards, e.g., SQL standard
 - ▶ Systematic (not a new construct per query query, no exceptions)
- ▶ **Orthogonality**
 - ▶ Possibility to freely combine query language constructs
 - ▶ Zero-One-Infinity principle (the only reasonable numbers in a programming language design are zero, one, and infinity)
- ▶ **Closed-form evaluation**
 - ▶ The result of a query is a proper object of the data model.

Comparison of Timestamps

- **Comparison of timestamps** is part of every temporal query language.
- Many query languages adopt (a variant of) **Allen's 13 period relations**:

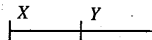
X before Y
Y after X



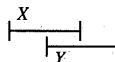
X equals Y



X meets Y
Y met_by X



X overlaps Y
Y overlapped_by X



X during Y
Y contains X



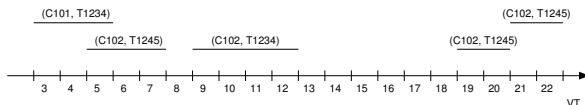
X starts Y
Y started_by X



X finishes Y
Y finished_by X



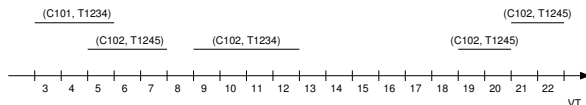
Video Rental Example/1



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

Q1: *All checkouts that overlap time period $[7, 9]$.*

Video Rental Example/1

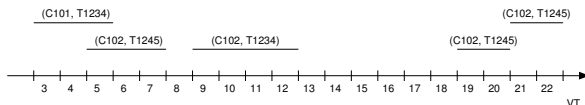


- Different queries on the Checkout table with their intended result:

Q1: *All checkouts that overlap time period [7, 9].*

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

Video Rental Example/1



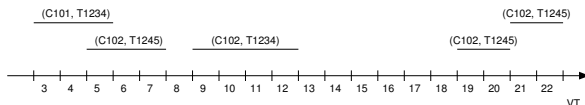
- Different queries on the Checkout table with their intended result:

Q1: *All checkouts that overlap time period [7, 9].*

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

Q2: *All 2-day checkouts.*

Video Rental Example/1



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

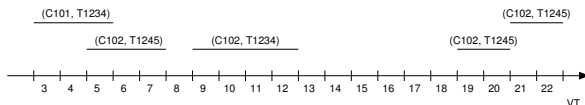
Q1: All checkouts that overlap time period [7, 9].

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

Q2: All 2-day checkouts.

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

Video Rental Example/1



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

Q1: All checkouts that overlap time period [7, 9].

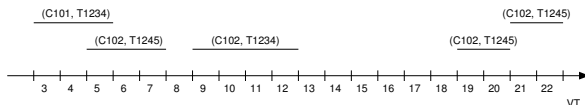
CustID	TapeNum	T
C102	T1245	[5, 7]
C102	T1234	[9, 12]

Q2: All 2-day checkouts.

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

Q3: At each time point which tapes were checked out together with tape T1234?

Video Rental Example/1



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

Q1: All checkouts that overlap time period [7, 9].

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

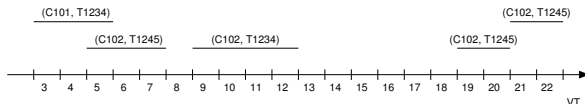
Q2: All 2-day checkouts.

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

Q3: At each time point which tapes were checked out together with tape T1234?

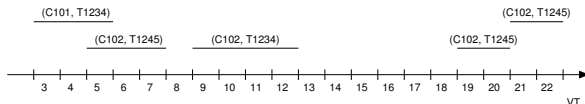
TapeNum	T
T1245	[5, 5]

Video Rental Example/2



Q4: *At each time point what is the number of tapes that have been checked out?*

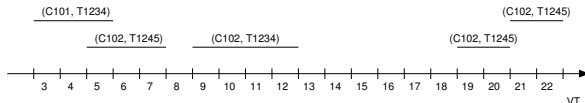
Video Rental Example/2



Q4: *At each time point 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]
⋮	⋮
⋮	⋮

Video Rental Example/2

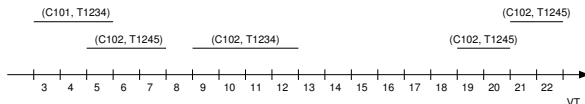


Q4: *At each time point what is the number of tapes that have been checked out?*

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

Q5: *How many checkouts were made in total?*

Video Rental Example/2



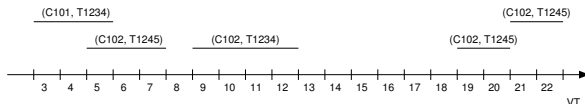
Q4: *At each time point what is the number of tapes that have been checked out?*

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

Q5: *How many checkouts were made in total?*

Cnt
5

Video Rental Example/2



Q4: At each time point what is the number of tapes that have been checked out?

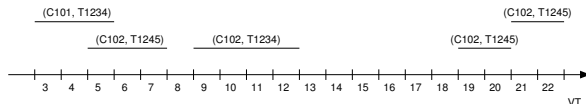
Cnt	T
1	[3,4]
2	[5,5]
1	[6,7]
0	[8,8]
1	[9,12]
0	[13,18]
⋮	⋮
⋮	⋮

Q5: How many checkouts were made in total?

Cnt
5

Q6: List all (current) checkouts (assume current time = 5).

Video Rental Example/2



Q4: At each time point 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 checkouts were made in total?

Cnt
5

Q6: List all (current) checkouts (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 is well understood technically.
- ▶ We assume **period timestamped** relations (or separate start and end points, which does not change much)

SQL + Abstract Data Types/2

CheckOut

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

SQL + Abstract Data Types/2

Q1 `SELECT *`
`FROM CheckOut`
`WHERE T OVERLAPS [7,9]`

CheckOut

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

SQL + Abstract Data Types/2

Q1 `SELECT *`
`FROM CheckOut`
`WHERE T OVERLAPS [7,9]`

Q2 `SELECT TapeNum, T`
`FROM CheckOut`
`WHERE DURATION(T) = 2`

CheckOut

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

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`

CheckOut

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

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`

Q4 too complicated

CheckOut

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

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`

Q4 too complicated

Q5 `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]

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`

Q4 too complicated

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 SQL + Abstract Data Types

- ▶ An abstract data type for periods is useful but 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 and complex solutions must be invented (= programmed) for queries that refer to “at each time point”.

IXSQL/1

- ▶ IXSQL extends SQL-92 with **(time) period** data type.
- ▶ Periods are convenient for representing temporal aspects, but create difficulties when formulating temporal queries
- ▶ IXSQL addresses this problem 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

CheckOut

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

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]`

CheckOut

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

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`

CheckOut

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

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]

IXSQL/3

- ▶ Unfold can be used to formulate queries about each time point.
- ▶ Idea of fold and unfold:
convert to time points; answer
query; convert to intervals

CheckOut

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

IXSQL/3

- ▶ Unfold can be used to formulate queries about each time point.
- ▶ Idea of fold and unfold:
convert to time points; answer
query; convert to intervals

Q4 `SELECT COUNT(*), T`
`FROM ( SELECT *`
`FROM CheckOut`
`REFORMAT AS UNFOLD T )`
`GROUP BY T`
`REFORMAT AS FOLD 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]

IXSQL/3

- ▶ Unfold can be used to formulate queries about each time point.
- ▶ Idea of fold and unfold:
convert to time points; answer
query; convert to intervals

Q4 `SELECT COUNT(*), T
FROM (SELECT *
FROM CheckOut
REFORMAT AS UNFOLD T)
GROUP BY T
REFORMAT AS FOLD 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]

- ▶ Without unfold we get regular SQL without any special processing of the temporal information.

IXSQL/3

- ▶ Unfold can be used to formulate queries about each time point.
- ▶ Idea of fold and unfold:
convert to time points; answer
query; convert to intervals

Q4 `SELECT COUNT(*), T`
`FROM ( SELECT *`
`FROM CheckOut`
`REFORMAT AS UNFOLD T )`
`GROUP BY T`
`REFORMAT AS FOLD 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]

- ▶ Without unfold we get regular SQL without any special processing of the temporal information.

Q5 `SELECT COUNT(*)`
`FROM CheckOut`

IXSQL/3

- Unfold can be used to formulate queries about each time point.
- Idea of fold and unfold:
convert to time points; answer
query; convert to intervals

Q4 `SELECT COUNT(*), T
FROM (SELECT *
FROM CheckOut
REFORMAT AS UNFOLD T)
GROUP BY T
REFORMAT AS FOLD 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]

- Without unfold we get regular SQL without any special processing of the temporal information.

Q5 `SELECT COUNT(*)
FROM CheckOut`

Q6 `SELECT CustID, TapeNum
FROM CheckOut
WHERE T OVERLAPS [NOW]`

Assessment of IXSQL

- ▶ Only two functions, fold and unfold, are added to SQL. This is attractive.
- ▶ Unfold can be used when needed to formulate queries about each time point (it is optional and not an invasive change).
- ▶ **Efficient evaluation** of queries formulated using fold/unfold has yet to be resolved.
- ▶ Neither a purely **point-based** nor **period based** view:
 - ▶ Sensitive to specific period representation of data (e.g., queries that do not use fold/unfold)
 - ▶ 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)
 - ▶ The combination of “at each time point” and periods is not supported (cf. temporal difference example for SQL/TP)

SQL/TP

- ▶ A **point-based** temporal extension of SQL
- ▶ Defined on temporal relations that use point timestamps
- ▶ Intervals do not exist at the logical level. They are used
 - ▶ internally (compact representation, efficient processing)
 - ▶ externally (presentation to users)
- ▶ Simple, unambiguous, and **well-defined semantics**

Q1 must compare neighboring DB states

```
SELECT a.*  
FROM CheckOut a, CheckOut b  
WHERE a.SeqNo = b.SeqNo  
AND (b.T = 7 OR b.T = 8 OR b.T = 9)
```

Q2 can only be answered if SeqNo is present. Idea is to use aggregation.

```
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

- Strength of SQL/TP is to extend snapshot queries to temporal queries if only a single state is involved. Idea is to extend snapshot queries with equality constraints on timestamps.

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

- Strength of SQL/TP is to extend snapshot queries to temporal queries if only a single state is involved. Idea is to extend snapshot queries with equality constraints on timestamps.

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`

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

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

- ▶ An additional attribute is needed for time-invariant aggregation.
 - ▶ The following fails for Q5:
 - ▶ `SELECT COUNT(*) FROM CheckOut`

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

- ▶ An additional attribute is needed for time-invariant aggregation.
 - ▶ The following fails for Q5:
 - ▶ `SELECT COUNT(*) FROM CheckOut`

Q5 `SELECT COUNT(DISTINCT SeqNo)
FROM CheckOut`

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

- ▶ An additional attribute is needed for time-invariant aggregation.
 - ▶ The following fails for Q5:
 - ▶ `SELECT COUNT(*) FROM CheckOut`

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

- ▶ **Additional attribute needed** to preserve information about periods of the argument relations.
- ▶ Additional attribute is **not a systematic approach** to obtain point-based semantics **and** associate information with 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 checkout periods are returned because the sequence numbers are not included
- ▶ Including the sequence number “disables” the difference

Assessment of SQL/TP

- ▶ **Period representation is still needed** for user interaction and internal representation:
 - ▶ 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).

TSQL2/1

- ▶ Comprehensive extension of SQL2 to support **time periods** 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.
- ▶ 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

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]

TSQL2/2

Q1 `SELECT *`
`FROM CheckOut`
`WHERE VALID(T) OVERLAPS period '7-9'`

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]

TSQL2/2

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]

Q1 `SELECT *`
`FROM CheckOut`
`WHERE VALID(T) OVERLAPS period '7-9'`

Q2 `SELECT SeqNo, TapeNum, T`
`FROM CheckOut`
`WHERE CAST(VALID(T) AS INTERVAL) = 2`

Result in Q2 must include SeqNo, otherwise value-equivalent tuples are coalesced.

TSQL2/2

Q1 `SELECT *`
`FROM CheckOut`
`WHERE VALID(T) OVERLAPS period '7-9'`

Q2 `SELECT SeqNo, TapeNum, T`
`FROM CheckOut`
`WHERE CAST(VALID(T) AS INTERVAL) = 2`

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]

Result in Q2 must include SeqNo, otherwise value-equivalent tuples are coalesced.

Q3 Implicit default valid clause in Q3

- ▶ `valid intersect(valid(a), valid(b))`
- ▶ i.e., non intersecting tuples do not contribute to the result

```
SELECT b.TapeNum
FROM CheckOut AS a, CheckOut AS b
WHERE a.TapeNum = T1234 AND b.TapeNum <> T1234
```

TSQL2/3

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 not well defined (what should default valid clause be?!)

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 not well defined (what should default valid clause be?!)

Q5 `SELECT SNAPSHOT COUNT(*)
FROM CheckOut`

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

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 not well defined (what should default valid clause be?)

Q5 `SELECT SNAPSHOT COUNT(*)
FROM CheckOut`

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

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/1

- ▶ ATSQL introduces **temporal statement modifiers** to add temporal support to SQL
- ▶ Statement modifiers are **semantic defaults** that indicate “at each time point” without specifying how to compute it.
- ▶ Provides a **systematic way** to construct temporal queries from nontemporal queries:
 1. Formulate the corresponding nontemporal query
 2. 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:
 - ▶ Periods have a meaning beyond a set of points

ATSQL/2

- ▶ 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 boundaries (change points) are 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]

ATSQL/2

- ▶ 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 boundaries (change points) are preserved.
- ▶ The rest of the query is standard SQL

Q1 `SEQ VT`

```
SELECT CustID, TapeNum, T
FROM CheckOut
WHERE T OVERLAPS [7,9]
```

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/2

- ▶ 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 boundaries (change points) are 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 CustID, TapeNum, T
FROM CheckOut
WHERE T OVERLAPS [7,9]
```

Q2 `SEQ VT`

```
SELECT TapeNum
FROM CheckOut
WHERE DURATION(T) = 2
```

ATSQL/3

- “At each point in time” is easy to specify and the specification is independent of the query that shall be evaluated at each time point.

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/3

- “At each point in time” is easy to specify and the specification is independent of the query that shall be evaluated at each time point.

CheckOut

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

Q3 **SEQ VT**
SELECT b.TapeNum
FROM CheckOut **AS** a, CheckOut **AS** b
WHERE a.TapeNum = T1234
AND b.TapeNum <> T1234

ATSQL/3

- “At each point in time” is easy to specify and the specification is independent of the query that shall be evaluated at each time point.

CheckOut

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

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

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

CheckOut

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

- ▶ 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

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

CheckOut

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

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

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

CheckOut

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

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`

- ▶ Language mechanism is **independent of the syntactic complexity** of the queries.
- ▶ The temporal parts are to a large degree separated from the nontemporal parts of the query.
- ▶ The semantics of SQL extended with statement modifiers has been defined.
- ▶ Represents a **more fundamental change** to the language design than other approaches, e.g., abstract data types.

Summary/1

- ▶ Temporal data models choose
 - ▶ time dimensions (valid time, transaction time, ...)
 - ▶ timestamps (points, intervals, temporal elements)
 - ▶ scope of timestamps (attribute, tuple, relation, ...)
- ▶ No single best choice exist. Assessment based on chosen timestamps is difficult; must be considered together with query language.
- ▶ There is no agreement whether periods should have a meaning beyond a set of points (most likely they should).

Summary/2

- ▶ There are a number of different temporal data models:
 - ▶ Snodgrass' tuple timestamping model
 - ▶ Jensen's backlog model
 - ▶ Ben-Zvi's tuple timestamped model
 - ▶ Gadia's attribute value timestamped model
- ▶ Language design criteria: expressive power, orthogonality, closedness, clarity, consistency
- ▶ Comparison of periods (Allen's predicates): convenient but not a major issue
- ▶ Now is a continuously moving constant
 - ▶ Understand the meaning of timestamps that include now
 - ▶ Often the largest timestamp is used to represent now but this leads to subtle problems (duration, etc)

Summary/3

- ▶ Concrete SQL-based query languages
 - ▶ **SQL + abstract data types**: extend SQL with abstract data type; gain is limited
 - ▶ **IXSQL**: two operators to convert periods to points and vice versa; simple and attractive; issues with performance and semantics
 - ▶ **SQL/TP**: only points exist; periods are used internally and for display; based on points, which gives simple semantics; issue with infinity, and presentation
 - ▶ **TSQL2**: use syntactic defaults to easily formulate temporal statements; comprehensive language; no precise semantics
 - ▶ **ATSQL**: use semantic defaults to easily formulate temporal statements; adds modifiers to SQL, which is a significant change