Teil I: Temporale DBS

1. Einführung

Gegenstand:

in DBMS und DB-Sprache eingebaute Unterstützung

der Speicherung und Anfragebearbeitung

von zeitbehafteten Daten aus Vergangenheit, Gegenwart, evtl. Zukunft

Applications

- Academic

  Transcripts record courses taken in previous and the current semester or term and grades for previous courses

- Accounting

  What bills were sent out and when, what payments were received and when?

  Delinquent accounts, cash flow over time

  Money-management software such as Quicken® can show e.g., account balance over time.

- Budgets

  Previous and projected budgets, multi-quarter or multi-year budgets

Applications, cont.

- Data Warehousing

  Historical trend analysis for decision support

- Financial

  Stock market data

  Audit analysis: why were financial decisions made, and with what information available?

  Program trading

- Geographic Information Systems (GIS)

  Land use over time: boundary of parcels change over time, as parcels get partitioned and merged.

  Title searches

Applications, cont.

- Insurance

  Which policy was in effect at each point in time, and what time periods did that policy cover?

- Inventory management

  Inventory over time, for time-series analysis, accounting

- Legal records

  Validity periods for laws

- Medical records

  Patient records, drug regimes, lab tests

  Tracking course of disease

  Longitudinal studies

Applications, cont.

- Payroll

  Past employees, employee salary history, salaries for future months, records of with-holding requested by employees

- Planning

  Use current and previous schedules for designing new schedules

  Planning for future contingencies

  Network management

- Process monitoring

  Chemical, nuclear power plant monitoring

  Active databases

Applications, cont.

- Project scheduling

  Milestones, task assignments

- Reservation systems

  E.g., airlines, trains, hotels

  Configuring new routes and offers
  to ensure high utilization

- Scientific applications

  Recording physical experiments

  Timestamping weather satellite images

  Dating archeological finds

Applications: Conclusion

- It is difficult to identify applications that do *not* involve time aspects and the management of temporal data.

- These applications would benefit from built-in temporal support in the DBMS.
  - More efficient application development
  - Potential increase in performance

An SQL Case Study

- University of Arizona’s Office of Appointed Personnel has some information in a database.

  Schema: Employee(Name, Salary, Title)

- Finding an employee’s salary is easy.

```
SELECT Salary
FROM   Employee
WHERE Name = 'Bob'
```

- The OAP wishes to add the date of birth

  Employee(Name, Salary, Title, DateofBirth **DATE**)

An SQL Case Study, cont.

- Finding the employee's date of birth is equally easy.

```
SELECT DateofBirth
FROM   Employee
WHERE  Name = 'Bob'
```

- Now the OAP wishes to computerize the employment history.

Employee (Name, Salary, Title, DateofBirth,
Start **DATE**, Stop **DATE**)

Store for each tuple the time interval [..) when it was valid.

An SQL Case Study, cont.

- To the data model, these new columns are structurally identical to DateofBirth.

Name	*Salary*	*Title*	*DateofBirth*	*Start*	*Stop*
Bob	60000	Assistant Provost	1945-04-19	1993-01-01	1993-06-01
Bob	70000	Assistant Provost	1945-04-19	1993-06-01	1993-10-01
Bob	70000	Provost	1945-04-19	1993-10-01	1994-02-01
Bob	70000	Professor	1945-04-19	1994-02-01	1995-01-01

- To find the employee's current salary, things are more difficult.

```
SELECT Salary
FROM Employee
WHERE Name = 'Bob'
    AND Start <= CURRENT_TIMESTAMP
    AND CURRENT_TIMESTAMP < Stop
```

An SQL Case Study, cont.

- OAP wants to distribute to all employees their salary history.

Output: For each person, maximal intervals for each salary are expected.

Name	*Salary*	*Start*	*Stop*
Bob	60000	1993-01-01	1993-06-01
Bob	70000	1993-06-01	1995-01-01

But: Employee could have arbitrarily many title changes between salary changes.

Thus: Consecutive value-equivalent tuples have to be **coalesced.**

An SQL Case Study, cont.

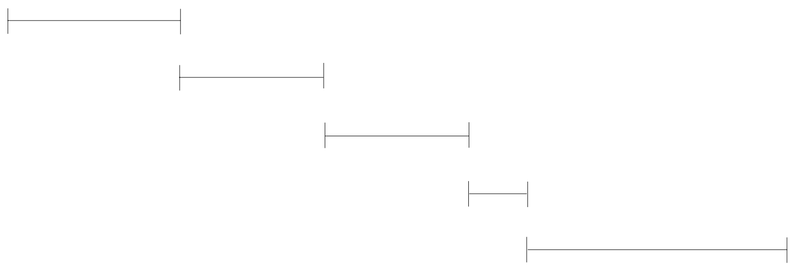
- Alternative 1

  Give the user a printout of Salary and Title information, and have user determine when his/her salary changed.

- Alternative 2

  Use SQL as much as possible.

  Find those intervals that overlap or are adjacent and thus should be merged.

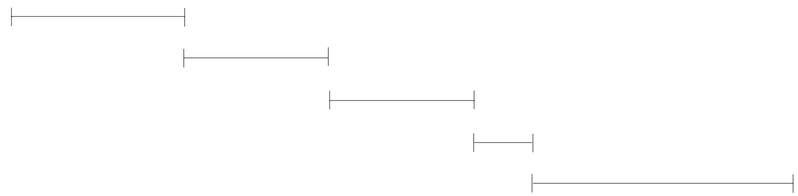


SQL Program

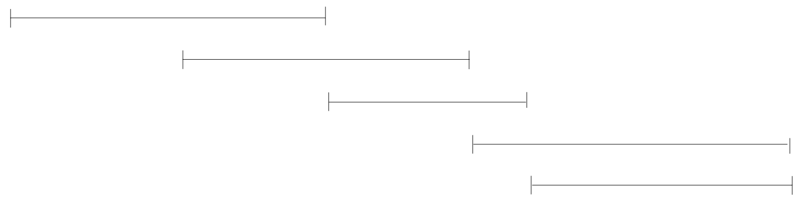
```
CREATE TABLE Temp(Salary, Start, Stop)
AS SELECT Salary, Start, Stop
FROM Employee
WHERE Name = 'Bob';

repeat
    UPDATE Temp AS T1
    SET (T1.Stop) = (SELECT MAX(T2.Stop)
                     FROM Temp AS T2
                     WHERE T1.Salary = T2.Salary
                           AND T1.Start < T2.Start
                           AND T1.Stop >= T2.Start
                           AND T1.Stop < T2.Stop)
     WHERE EXISTS (SELECT *
                     FROM Temp AS T2
                     WHERE T1.Salary = T2.Salary
                           AND T1.Start < T2.Start
                           AND T1.Stop >= T2.Start
                           AND T1.Stop < T2.Stop)
 until no tuples updated;
```

Example

- Initial table



- After first iteration:



- After second iteration:

SQL Program, cont.

- The loop is executed $log_2 N$ times in the worst case, where N is the number of tuples in a chain of overlapping or adjacent, value-equivalent tuples.
- Then delete extraneous, non-maximal intervals.

```
DELETE FROM Temp T1
WHERE EXISTS (SELECT *
              FROM Temp AS T2
              WHERE T1.Salary = T2.Salary
                    AND ((T1.Start > T2.Start
                          AND T1.Stop <= T2.Stop)
                      OR (T1.Start >= T2.Start
                          AND T1.Stop < T2.Stop))
```

[0] Die SQL-Anweisung ist allgemeiner als hier nötig angelegt.

• Alternative 3: Use pure SQL.

```
CREATE TABLE Temp(Salary, Start, Stop)
AS SELECT Salary, Start, Stop
     FROM Employee
     WHERE Name = 'Bob';

SELECT DISTINCT F.Salary, F.Start, L.Stop    // teste First/Last-Paar
FROM Temp AS F, Temp AS L
WHERE F.Start < L.Stop
     AND F.Salary = L.Salary
     AND NOT EXISTS (SELECT *                          // keine Lücken?
                    FROM Temp AS M
                    WHERE M.Salary = F.Salary
                         AND F.Stop < M.Start
                         AND M.Start < L.Stop
                         AND NOT EXISTS (SELECT *
                              FROM Temp AS T1
                              WHERE T1.Salary = F.Salary
                                   AND T1.Start < M.Start
                                   AND M.Start <= T1.Stop))
     AND NOT EXISTS (SELECT *                          // maximal?
                    FROM Temp AS T2
                    WHERE T2.Salary = F.Salary
                         AND ((T2.Start < F.Start
                                   AND F.Start <= T2.Stop)
                              OR (T2.Start <= L.Stop
                                   AND L.Stop < T2.Stop)))
```

Using More Procedural Code

- Alternative 4: Use SQL only to open a cursor on the table.

Check subsequent intervals in an ordered list, whether they can be coalesced. Replace them by the new intervals.

```
DECLARE emp_cursor CURSOR FOR
        SELECT Salary, Start, Stop FROM Employee
        WHERE Name='Bob' ORDER BY Salary,Start;
OPEN emp_cursor;

loop:
      FETCH emp_cursor INTO :salary,:start,:stop;
      if no data returned then go to finished;
      <compare start/stop-values with previous ones ...>
      go to loop;

finished:
CLOSE emp_cursor;
```

How the query should look like

in a temporal query language:

```
TEMPORALLY SELECT Salary
FROM Employee
WHERE Name = 'Bob'
```

An SQL Case Study, cont.

- Drastic Alternative: Decompose the schema

  Separate Salary and Title information; store coalesced intervals

  Employee1 (Name, Salary, Start **DATE**, Stop **DATE**)

  Employee2 (Name, Title, Start **DATE**, Stop **DATE**)

- Getting the salary information is now easy:

  SELECT Salary, Start, Stop
  FROM Employee1
  WHERE Name = ’Bob’

- But what if the OAP wants a table of (salary, title)-intervals?

An Example of a Temporal Join

- Employee1:

Name	*Salary*	*Start*	*Stop*
Bob	60000	1993-01-01	1993-06-01
Bob	70000	1993-06-01	1995-01-01

- Employee2:

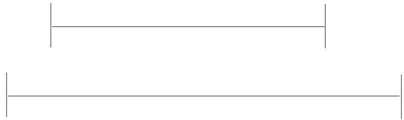
Name	*Title*	*Start*	*Stop*
Bob	Assistant Provost	1993-01-01	1993-10-01
Bob	Provost	1993-10-01	1994-02-01
Bob	Full Professor	1994-02-01	1995-01-01

- Result of temporal join:

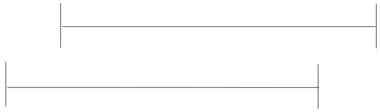
Name	*Salary*	*Title*	*Start*	*Stop*
Bob	60000	Assistant Provost	1993-01-01	1993-06-01
Bob	70000	Assistant Provost	1993-06-01	1993-10-01
Bob	70000	Provost	1993-10-01	1994-02-01
Bob	70000	Full Professor	1994-02-01	1995-01-01

Temporal Join in SQL

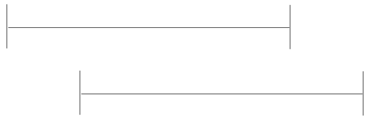
```
SELECT Employee1.Name, Salary, Title,
                    Employee1.Start, Employee1.Stop
FROM Employee1, Employee2
WHERE Employee1.Name = Employee2.Name
     AND Employee2.Start <= Employee1.Start
     AND Employee1.Stop <= Employee2.Stop
```



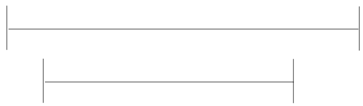
```
UNION
SELECT Employee1.Name, Salary, Title,
                    Employee1.Start, Employee2.Stop
FROM Employee1, Employee2
WHERE Employee1.Name = Employee2.Name
     AND Employee1.Start > Employee2.Start
     AND Employee2.Stop < Employee1.Stop
     AND Employee1.Start < Employee2.Stop
```



```
UNION
SELECT Employee1.Name, Salary, Title
                    Employee2.Start, Employee1.Stop
FROM Employee1, Employee2
WHERE Employee1.Name = Employee2.Name
      AND Employee2.Start > Employee1.Start
      AND Employee1.Stop <= Employee2.Stop
      AND Employee2.Start < Employee1.Stop
```



```
UNION
SELECT Employee1.Name, Salary, Title
                    Employee2.Start, Employee2.Stop
FROM Employee1, Employee2
WHERE Employee1.Name = Employee2 Name
      AND Employee2.Start => Employee1.Start
      AND Employee2.Stop <= Employee1.Stop
```



How the query should look like

in a temporal query language:

```
TEMPORALLY SELECT
      Employee1.Name, Salary, Title
FROM Employee1, Employee2
WHERE
      Employee1.Name = Employee2.Name
```

Summary

- Applications manage temporal data.

- If a temporal DBMS is used

  Schemas are simpler (temporal table)

  SQL queries are simpler (temporally select)

  Much less procedural code is needed

- Benefits

  Application code is less complex

    Easier to understand
    Easier to ensure correctness
    Easier to maintain

  Performance may be enhanced by
  delegating functionality to the DBMS.

2. Zeitmodelle

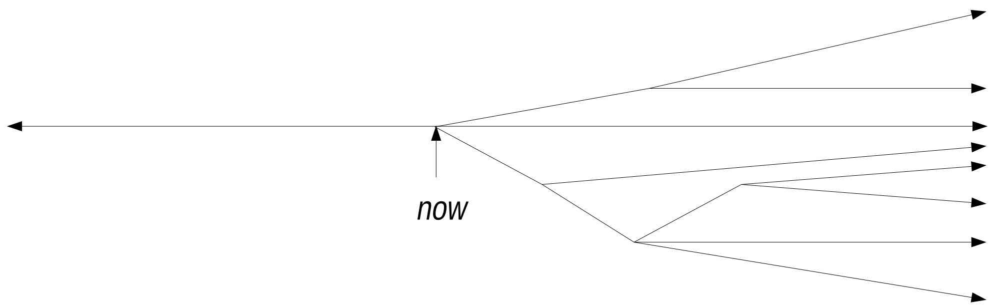
Time Structure

- *Linear*

We will assume linear time.

Other Time Structures

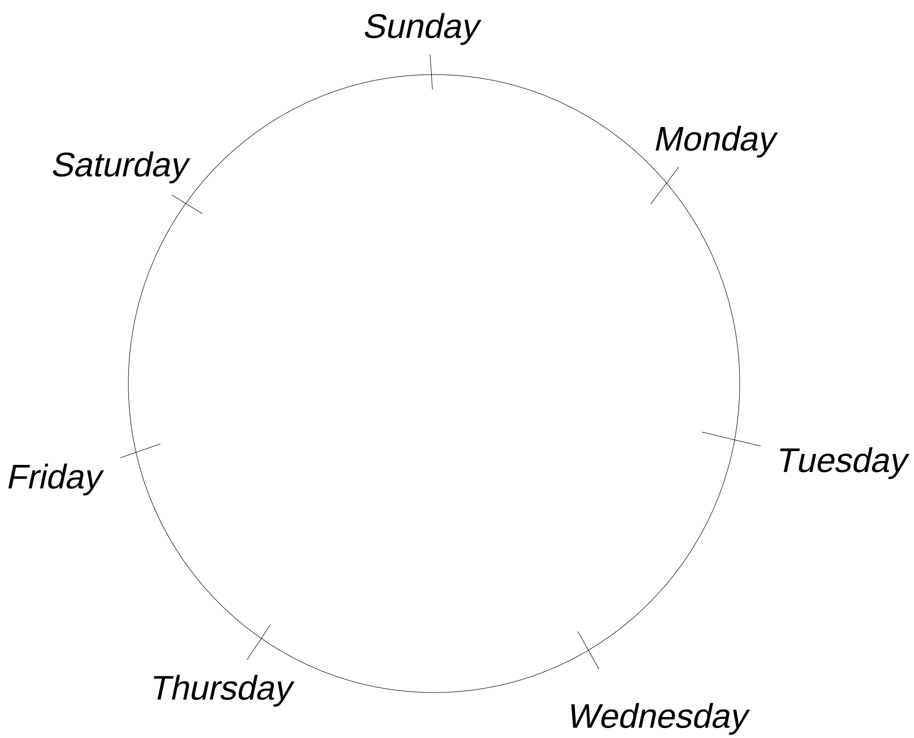
- *Possible (Hypothetical) Futures*



- *Tree = Branching Time*
- *or Directed Acyclic Graph*

Other Time Structures, cont.

- *Periodic/Cyclic Time*



Time Density

- *Discrete*
  - Time line is isomorphic to the integers.
  - Time line composed of a sequence of non-decomposable time periods of some fixed minimal duration, termed *chronons*.
  - Between each pair of instants=chronons is a finite number of other instants.
- *Dense*
  - Time line is isomorphic to the rational numbers.
  - Between each pair of instants is an infinite number of other instants.

Time Density, cont.

- *Continuous*
  - Time line is isomorphic to the real numbers.
  - Between each pair of instants is an infinite number of other instants.

- Temporal DBS work with discrete time.

Clocks

- A *clock* is a physical process coupled with a method of measuring that process.
- The units of measurement are reported in terms of chronons of the clock.
- Their *granularity* may depend on application, e.g. seconds, days, weeks, months.
- Clocks (or *calendars*) provide the semantics of a timestamp representation.

Temporal Data Types

- A time *instant t* is a point on the time line. *[dt. Zeitpunkt]*
  - An *event* is an instantaneous fact, i.e, something occurring at an instant. The *event occurrence time* of an event is the instant at which the event occurs in the real world.

- An *instant set* is a set of instants.

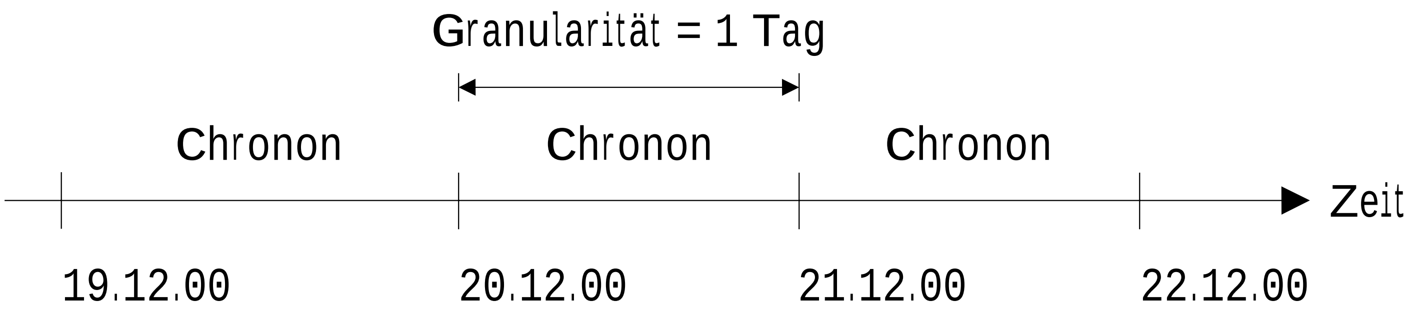
- A *time period* $[t_1,t_2)$ is the time between two instants. *[dt. Intervall]*
  - Also called *interval*, but this differs from the SQL data type **`INTERVAL`**, e.g."3 years".

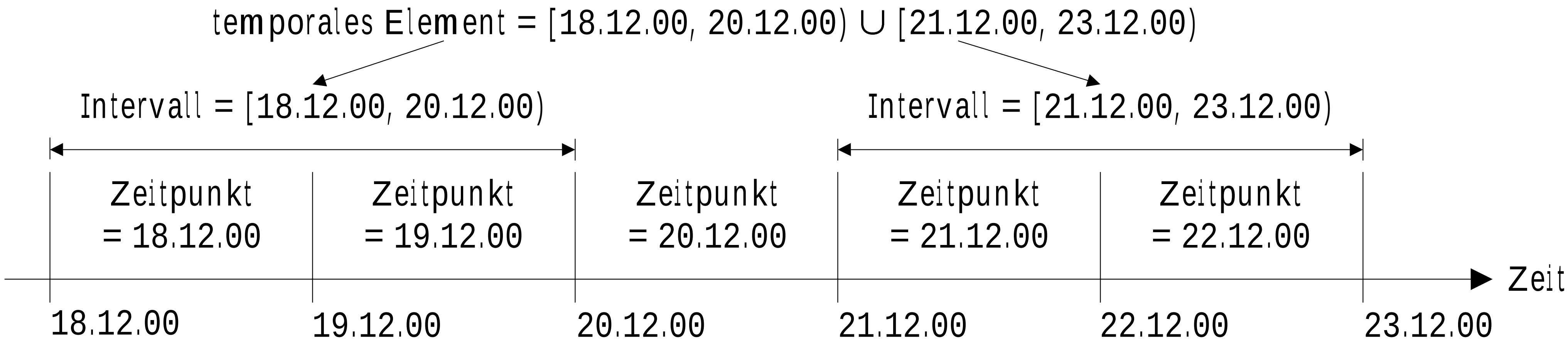
Temporal Data Types, cont.

- A *time span* is a directed duration of time. A *duration* is an amount of time with a known length, but no specific starting or ending instants. *[dt. Zeitspanne]*
    - A *positive span* denotes forward motion of time; a *negative span* denotes backwards motion of time.

- A *temporal element* is a finite union of intervals. *[dt. temporales Element]*

  The component intervals are usually considered to be disjoint and non-adjacent.


Granularität = 1 Tag
Chronon
Chronon
Chronon
Zeit
19.12.00
20.12.00
21.12.00
22.12.00


temporales Element = [18.12.00, 20.12.00) ∪ [21.12.00, 23.12.00)
Intervall = [18.12.00, 20.12.00)
Intervall = [21.12.00, 23.12.00)
Zeitpunkt = 18.12.00
Zeitpunkt = 19.12.00
Zeitpunkt = 20.12.00
Zeitpunkt = 21.12.00
Zeitpunkt = 22.12.00
Zeit
18.12.00
19.12.00
20.12.00
21.12.00
22.12.00
23.12.00

Temporal Dimensions

- *Valid time* of a fact: when the fact is true in the modeled reality
  [dt. Gültigkeitszeit]
- *Transaction time* of a fact: when the fact is current in the database and may be retrieved *[dt. Transaktionszeit]*

- Four kinds of tables instead of one kind:
  - *snapshot*
  - *valid-time*
  - *transaction-time*
  - *bitemporal*

Example: Tom’s Employment History

- On January 1, 1984, Tom joined the faculty as an Instructor.

- On December 1, 1984, Tom completed his doctorate, and so was promoted to Assistant Professor retroactively on July 1, 1984.

- On March 1, 1989, Tom was promoted to Associate Professor, proactively on July 1, 1989.

A Snapshot Table

- Can be modified
- Used for static queries, e.g.:

  What is Tom’s rank?

  SELECT Rank
  FROM Faculty
  WHERE Name = ’Tom’

Analogy: Doorplate

(Jan. 1984)

Instructor Tom

(Dec. 1984)

Assistant
Prof. Tom

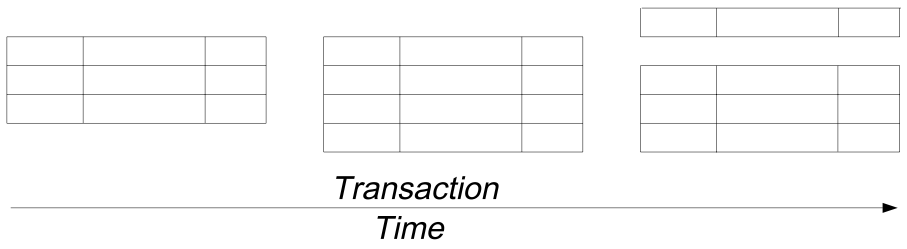
(March 1989)

Assistant
Prof. Tom

(July 1989)

Associate
Prof. Tom

A Transaction-time Table

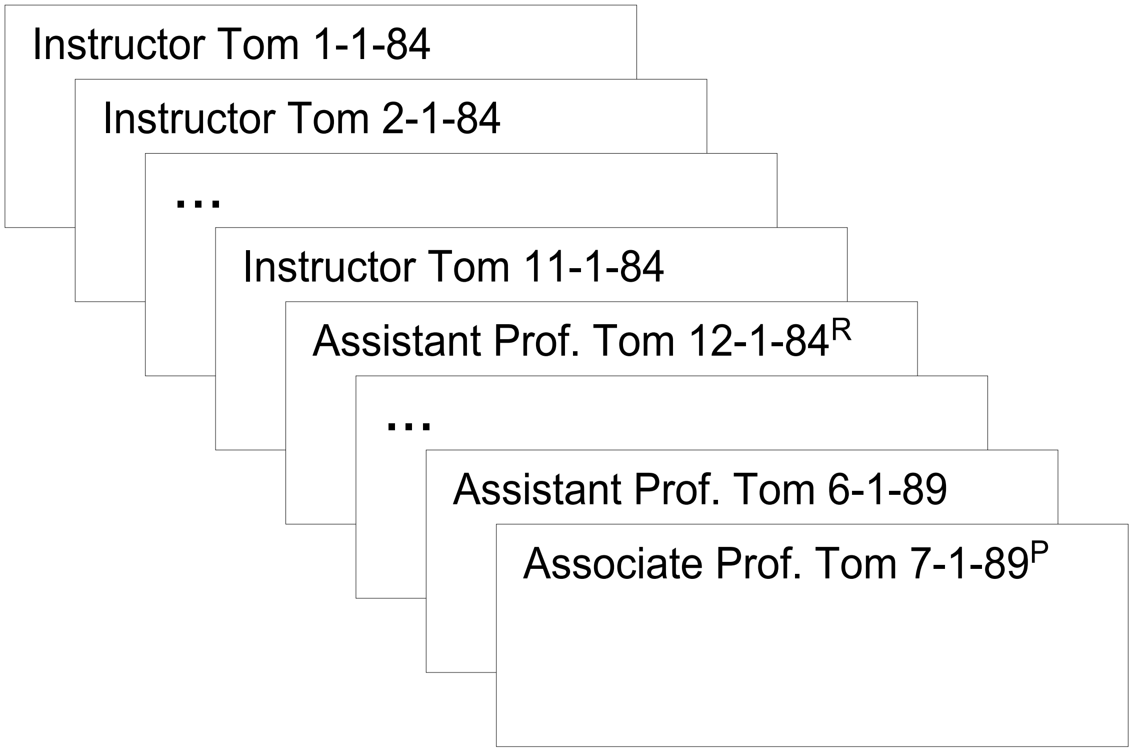


- Append-only: correction to previous snapshot states is not permitted
- Supports transaction time
- Supports rollback queries like:

What did we believe on Oct. 1, 1984 what Tom's rank was?

SELECT Rank
FROM Faculty
WHERE Name = 'Tom' **AND**
  TRANSACTION(Faculty) **OVERLAPS DATE** '1984-10-01'

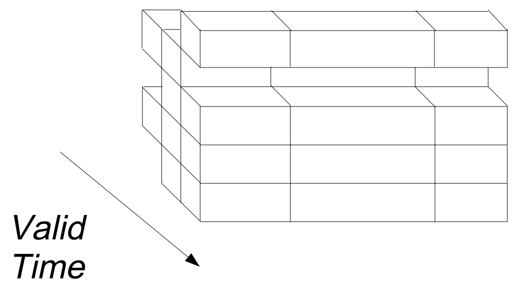
Analogy: Pay Stubs



R = incl. proof of retroactive payment for months 7...11-84

P = update event reserved externally on 3-1-89 (no future time handling!)

A Valid-Time Table



- May be modified
- Supports valid time
- Supports historical queries like:

  What was Tom’s rank on October 1, 1984 (as currently best known)?

SELECT Rank
FROM Faculty
WHERE Name = ’Tom’
AND VALID(Faculty) **OVERLAPS DATE** ’1984-10-01’

Analogy: A Vita

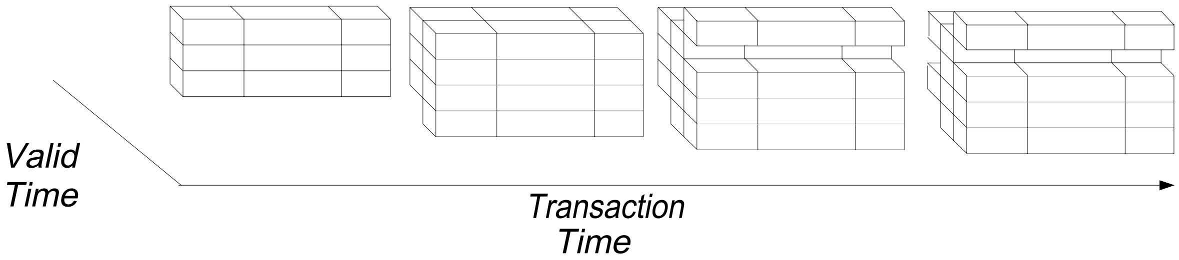
Tom

Employment

Associate Professor	July, 1989
Assistant Professor	July, 1984
Instructor	January, 1984

(as of March 1989 or later)

A Bitemporal Table



- Append-only
- Supports valid and transaction time
- Supports rollback coupled with historical queries like:
- On October 1, 1984, what did we think Tom's rank was at that date?

```
SELECT Rank
FROM   Faculty AS F
WHERE Name = 'Tom'
    AND VALID(F) OVERLAPS DATE '1984-10-01'
    AND TRANSACTION(F) OVERLAPS DATE '1984-10-01'
```

Analogy: A Stack of Vitæ

January 1984 version

Tom

Employment

Instructor	January, 1984

December 1984 version

Tom

Employment

Assistant Professor	July, 1984
Instructor	January, 1984

March 1989 version

Tom

Employment

Associate Professor	July, 1989
Assistant Professor	July, 1984
Instructor	January, 1984

Here, only valid/transaction start times are given; corresponding intervals are obvious.

Summary

- Several different structures of time
  - Linear is simplest and most common.
- Five fundamental temporal data types
- Many different physical clocks/calendars
- Several dimensions of time
  - Valid time
  - Transaction time