Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Chapter 10 – Temporal Data Models

Outline

Overview

I. Object-Relational Database Concepts

1. User-defined Data Types and Typed Tables
2. Object-relational Views and Collection Types
3. User-defined Routines and Object Behavior
4. Application Programs and Object-relational Capabilities

II. Online Analytic Processing

5. Data Analysis in SQL
6. Windowed Tables and Window Functions in SQL

III. XML

7. XML Data Modeling
8. XQuery
9. SQL/XML

IV. More Developments

10. **Temporal Data Models**

Outline

- Motivation
- Conceptual temporal data model
  - periods & Allen's operators
  - valid time, transaction time
- Temporal data management in SQL:2011
  - time periods
  - application-time period tables
  - system-versioned tables
- Summary

Motivation

- Databases are models of the real world
  - focus is usually on the current state: updates change the DB to reflect current state of the real world
  - information about the previous state(s) is no longer available
- Consider the following application scenarios
  - for auditing purposes, a bank needs to keep logs of client record changes for the last 5 years
  - for internal accounting, a company needs to track the department membership times of an employee and to which projects she was contributing during which time periods
  - a travel agency wants to detect inconsistencies in travel itineraries, e.g., booking a hotel in New York for 8 days and a rental car in Rome for some of those days doesn't make sense

A Common "Temporalization" Approach

- Use a timestamp-based modelling approach
  - one additional column to represent since when the fact represented by a tuple was true (or stored in the DB)
    - now we only know since when the current state holds and it is assumed to still hold
    - no record about past states, unless we "compute" it from next-higher timestamp of another, value-equivalent tuple
    - no old state for deleted tuples (e.g., employee has left the company)
  - two additional columns to represent the time period during which the fact was true/stored
    - how do we interpret the timestamps (included or excluded in the time period)?
    - how do we issue temporal queries? complex!
    - how can we avoid redundancies, i.e., two value-equivalent tuples with overlapping time periods?
    - how can we avoid conflicts, i.e., two "versions" of the same "object" with overlapping time periods? primary keys are "broken"! E.g., PK(empno, start, stop) doesn't help!

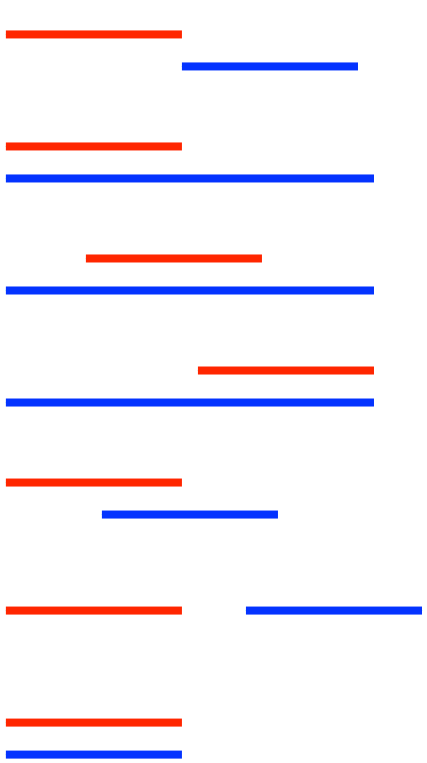
      example: EMP(emp1, dept1, day1, day7), EMP(emp1, dept2, day5, day6) ???
- Adding real temporal support "on top of the DBMS" by using triggers, constraints, stored procedures is expensive, if not impossible!

Time Periods

- Definition: A period is a (mathematical) interval on the timeline, characterized by start and end times
  - start time and/or end time may be included or excluded from the period
  - Notation: [a, b], [a, b), (a, b], (a, b)
- Quantisation of time
  - we think of time as being continuous
  - quantisation turns a period into a finite set of discrete points (or "chronons")
  - the distance between the points is according to a chosen scale (e.g., days, minutes, seconds, ...)
  - each point has a successor (next) and a predecessor (prior) point
  - bound points of a period: pre, begin, end, post
  - equivalence: [begin, end] = (pre, end] = [begin, post) = (pre, post)
- Example:
  - scale is days, we're using integers in our example (d01, d02, ...)
  - [d02, d09] = (d01, d09] = [d02, d10) = (d01, d10)

Operators on Time Periods

- Membership test of point in period
- Union, intersection, minus, count
  - union, intersection are not defined for all periods! Why?
- Comparison predicates: Allen's operators (are not symmetric!)
  - p1 meets p2
  - p1 starts p2
  - p1 during p2
  - p1 finishes p2
  - p1 overlaps p2
  - p1 before p2
  - p1 = p2



Conceptual Model for Temporal Relations

- Tuples in a "classic" relation represent current state information
- We extend the tuples by adding information about a time period
  - allows to keep historic information, track changes, represent validity periods, etc.
  - tuples are called value-equivalent, if they have the same values for all columns, ignoring the time period
  - a snapshot relation is obtained by selecting for a given point in time the tuples for which the point is contained in the associated time period
  - goal: a snapshot relation should meet all the properties of a "classic" relation (primary keys, foreign key constraints have to hold!)
- Example:

ENo	EDept	Start	End
22217	3	d01	d04
22217	4	d05	d07
22217	3	d08	d012

d03-snapshot

ENo	EDept
22217	3

Valid Time vs. Transaction Time

- What is the semantics of the time period?
- Tables with valid time semantics (also called "application time")
  - time period represents temporal validity information that is defined and interpreted by the application of application domain
    - department membership period, period of marriage, coverage period for an insurance policy, etc.
  - application specifies valid time period during insert, update, deletion of tuples
- Tables with transaction time semantics (also called "system time")
  - time period represents the time during which a tuple was "current" in the database
    - tracks versions of tuples using a transaction-specific timestamp, keep history for auditing
  - DBMS is in charge of maintaining time periods of the tuples and possibly insert new tuples to reflect tuple updates
  - the application cannot modify time periods
- Bitemporal tables
  - tuples are associated with both, a valid time period and a transcation time period

SQL:2011 – Temporal Support

- Time periods
  - supported through (named) period definition on a table
  - based on column pairs holding the start/end times
    - either of type DATE or TIMESTAMP (scale of the period), same datatype for both columns
    - uses a closed-open interval model – [start-col, end-col)
  - no new time period datatype added!
- System-versioned tables
  - provide transaction time support
  - queries by default range over current system rows
  - FOR SYSTEM TIME clause allows for historical system row queries
- Application-time period tables
  - provide support for valid time
  - primary key and referential integrity support
  - queries range over all rows, predicates over periods allow for temporal queries
- Bitemporal tables are supported

Application-Time Period Tables

- User/application is in charge of setting/updating valid time periods
- CREATE/ALTER TABLE can specify an application-time period using the PERIOD FOR clause
  - Example

```
CREATE TABLE Emp(
   ENo       INTEGER,
   EStart    DATE,
   EEnd      DATE,
   EDept     INTEGER,
   PERIOD FOR EPeriod (EStart, EEnd))
```

  - arbitrary period name can be used
  - start and end columns are regular columns of the table
- INSERT statements now need to contain values for application-time period
  - Example

```
INSERT INTO Emp
VALUES (22217, DATE '2010-01-01',DATE '2011-11-12', 3)
```

Application-Time – Updates and Deletions

- Modifications can be specified for an effective time period
- Consider table Emp:
  - changes are applied on all rows with overlapping periods
  - rows may get split into two or three consecutive rows

```
UPDATE Emp
FOR PORTION OF EPeriod
      FROM DATE '2011-02-03'
      TO DATE '2011-09-10'
SET EDept = 4 WHERE ENo = 22217

DELETE Emp
FOR PORTION OF EPeriod
      FROM DATE '2011-02-03'
      TO DATE '2011-09-10'
WHERE ENo = 22217
```

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-11-12	3

UPDATE

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-02-03	2011-09-10	4
22217	2011-09-10	2011-11-12	3

DELETE

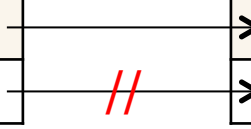
ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-09-10	2011-11-12	3

Application-Time & Primary/Foreign Keys

- Primary keys need to include the application-time period
  - otherwise there can be only one row for each entity with a single time period
  - just adding the start/end columns to the PK will not forbid overlaps!

  ALTER TABLE Emp ADD PRIMARY KEY (ENo, **EPeriod WITHOUT OVERLAPS**)
- Referential constraints can be generalized to hold for every valid point in time
  - Example where constraint does not hold for every point in time:

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4



DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2011-06-01	2011-12-31	QA

  - How to avoid this: include time period in both primary and foreign keys:

```
CREATE TABLE Dept
(DNo INTEGER, ..., PERIOD FOR DPeriod (DStart, DEnd),
PRIMARY KEY (DNo, DPeriod WITHOUT OVERLAPS))

ALTER TABLE Emp
ADD FOREIGN KEY (Edept, PERIOD EPeriod)
REFERENCES Dept (DNo, PERIOD DPeriod)
```

Query Support For Application Time

- The regular SELECT syntax is used
  - time period start/end columns can be queried in the usual way
- Additional support through special period predicates
  - Example (using CONTAINS for point membership):
    SELECT Ename, Edept
    FROM Emp
    WHERE ENo = 22217 AND EPeriod CONTAINS DATE '2011-01-02'
- Semantics of additional predicates (based on Allen's operators)
  - x CONTAINS y ~ (x contains y) or (x starts y) or (x finishes y) or (x equal y)
  - x OVERLAPS y ~ (x overlaps y) or (y overlaps x) or (x contains y) or (y contains x) or (x starts y) or (y starts x) or (x finishes y) or (y finishes x) or (x equals y)
  - x EQUALS ~ x equals y
  - x PRECEDES ~ (x before y) or (x meets y)
  - x SUCCEEDS y ~ (y before x) or (y meets x)
  - x IMMEDIATELY PRECEDES y ~ x meets y
  - x IMMEDIATELY SUCCEEDS y ~ y meets x

Temporal Joins Using Application-Time

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4

DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2009-06-01	2011-01-31	Quality
4	2011-02-01	2011-12-31	QA

- Without additional predicates, a join operation does not consider temporal aspects
  - Example: The following query return 3 rows

```
SELECT ENo, EDept, DName
FROM Emp, Dept
WHERE EDept = DNo
```

- For temporal joins, period predicates need to be used
  - Example: The following query return 2 rows

```
SELECT ENo, EDept, DName
FROM Emp, Dept
WHERE EDept = DNo AND EPeriod OVERLAPS DPeriod
```

System-Versioned Tables

- System-versioned tables support the notion of transaction time
  - any table that contains a period definition with the standard-specified name, SYSTEM_TIME, and includes the keywords WITH SYSTEM VERSIONING

    Example:

```
CREATE TABLE Emp (
ENo INTEGER,
Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
EName VARCHAR(30),
PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end))
WITH SYSTEM VERSIONING
```

- Only DBMS can (and has to) provide start and end times for the periods
  - standard does not defines what timestamp to use, only that it is the same for all operations within a transaction
- Any modification will result in an additional tuple that represents the old row state before the update is performed
  - current system row: a row whose system period contains the current timestamp
  - historical system row: any row that is not a current system row

Changing System-Versioned Tables

- INSERT
  - automatically sets the period start to the transaction timestamp and the period end to the highest data value supported for the type

```
INSERT INTO Emp (ENo, EName)
VALUES (22217, 'Joe')
```

ENo	Sys_start	Sys_end	EName
22217	2012-01-01 09:00:00	9999-12-31 23:59:59	Joe

- UPDATE and DELETE
  - only operate on the current row
  - will automatically insert a historical system row for every row that is updated or deleted
    - historical row is a copy of the current row with the period end set to the transaction timestamp
  - UPDATE implicitly sets the period start to the transaction timestamp

```
UPDATE Emp
SET EName = 'Tom'
WHERE ENo = 22217
```

ENo	Sys_start	Sys_end	EName
22217	2012-01-01 09:00:00	2012-02-03 10:00:00	Joe
22217	2012-02-03 10:00:00	9999-12-31 23:59:59	Tom

Constraints & Queries

- Primary key and foreign key constraints for system-versioned tables
  - only need to be enforced on the current system rows
  - system-time period does not need to be included in the key!
- Queries on system-versioned tables
  - by default only retrieves the current rows
  - can address historical rows using the FOR SYSTEM TIME clauses
  - select rows that were current as of a given timestamp:

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME AS OF TIMESTAMP '2011-01-02 00:00:00'
```

  - select rows that were current during a given time period, not including end TS

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME
FROM TIMESTAMP '2011-01-02 00:00:00' TO TIMESTAMP '2011-12-31 00:00:00'
```

  - select rows that were current during a given time period, including end TS

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME  BETWEEN TIMESTAMP '2011-01-02 00:00:00'
                              AND TIMESTAMP '2011-12-31 00:00:00'
```

Bitemporal Tables

- Application-time and system-versioned tables combined
  - results in the capabilities of both forms of temporal support
  - Example:

```
CREATE TABLE Emp(
ENo INTEGER, EStart DATE, EEnd DATE, EDept INTEGER,
PERIOD FOR EPeriod (EStart, EEnd),
Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
Sys_end TIMESTAMP(12) GENERATED  ALWAYS AS ROW END,
EName VARCHAR(30),
PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end),
PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS),
FOREIGN KEY  (Edept, PERIOD EPeriod) REFERENCES Dept (DNo, PERIOD DPeriod))
WITH SYSTEM VERSIONING
```

- Queries can specify predicates on both application-time and system-time periods
  - Example:

```
SELECT ENo, EDept
FROM Emp  FOR SYSTEM_TIME AS OF TIMESTAMP '2011-07-01 00:00:00'
WHERE ENo = 22217 AND  EPeriod CONTAINS DATE '2010-12-01'
```

Summary

- Modelling temporal data is an important requirement in many application areas
  - time periods, predicates – Allen's operators
  - valid and transaction time, bitemporal tables
- Important previous efforts include
  - TSQL2
  - SQL/Temporal
- SQL:2011 Capabilities
  - period definitions
  - application-time period tables for valid time support
  - system-versioned tables for transaction time support
  - combination – bitemporal tables
- SQL future directions
  - outer join, grouping/aggregation involving periods
  - normalization – collapsing contiguous, value-equivalent rows
  - multiple application-time periods per table

Literature

- K. Kulkarni, J.-E. Michels, "Temporal features in SQL:2011", in ACM SIGMOD Record 41(3), September 2012, pp. 34-43.
- J.F. Allen, "Maintaining knowledge about temporal intervals", Communications of ACM 26(11), November 1983.
- C. J. Date, H. Darwen, N.A. Lorentzos, "Temporal Data and the Relational Model", Morgan Kaufman, 2003.
- R.T. Snodgrass, et. al.," TSQL2 Language Specification". SIGMOD Record (SIGMOD) 23(1):65-86 (1994).