Temporal and Spatial Data Management
Fall 2019

Introduction
SL01

- Course setup and administration
- New requirements for database systems
- Motivation for temporal database functionality

Table of Contents

- **Course setup and administration**
  - **Lectures**
  - **Exercises**
  - **Exam**
  - **Literature**
  - **Databases**
- New requirements for database systems
- Motivation for temporal database functionality
  - What is a temporal database
  - The need for temporal databases
  - Case study

About me

- My employment history:

University	Where	When
ETH Zürich		[1990, 1994)
University of Arizona		[1994, 1995)
Aalborg University		[1995, 2003)
University of Bolzano		[2003, 2009)
University of Zürich		[2009, now)

- Since more than twenty years I have been an active researcher in the area of temporal, spatial and spatiotemporal database systems.

This Course/1

- Pre-conditions (what I assume you know/handle yourself):
  - Read, write and analyze SQL statements
  - Read, write and analyze relational algebra and calculus expressions
  - Algorithms (pseudocode)
  - Work with PostgreSQL
- What we do:
  - Work out concrete solutions (SQL, relational calculus, algorithms)
  - Reflect about solutions (identify and understand pros and cons)
- What is not sufficient:
  - General/abstract knowledge about the topic
  - Memorizing solutions without understanding details

This Course/2

- Course page
  - http://www.ifi.uzh.ch/dbtg
  - The course is taught every second year (odd years)
- Syllabus
  1. Introduction, new requirements, motivation
  2. Time domain, timestamps, granularity, calendar
  3. Temporal semantics
  4. Temporal data models and extensions of SQL
  5. Algorithms for temporal query processing
  6. Spatial databases
  7. Query processing in spatial network databases

This Course/3

- Location: BIN 2.A.01
- Time: Monday 14:00 - 15:45
- The course is an advanced course in the area of database systems.
- The course is research based. In some cases there is no consensus about the best approach. World is not black and white.
- The course material consists of the course slides, the in-class exercises, and selected research papers or book chapters.

This Course/4

- The **time-related problems** that we discuss are present in many applications
- The ANSI/ISO SQL:2011 Standard includes temporal features:
  https://files.ifi.uzh.ch/dbtg/tsdm/hs19/KM12.pdf
- Basic infrastructure for the time is being incorporated into commercial systems.
  - IBM DB2: implementation of SQL standard
    https://files.ifi.uzh.ch/dbtg/tsdm/hs19/Nic12.pdf
  - Oracle: Workspace Database Manager
    https://files.ifi.uzh.ch/dbtg/tsdm/hs19/OWMpartI.pdf
  - Teradata: current, sequenced, and non-sequenced statements
    https://files.ifi.uzh.ch/dbtg/tsdm/hs19/AGC+13.pdf
  - PostgreSQL: extension that supports temporal tables
    https://github.com/arkhipov/temporal_tables
  - Microsoft SQL Server: system-versioned tables
    https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables

This Course/5

- A number of database systems support **spatial functionality** (spatial measurements, spatial functions, spatial predicates, geometry constructors, observer functions):
  - **PostGIS**: spatial extension of PostgreSQL that implements the standardized datatype geometry and corresponding functions.
  - **Oracle Spatial**: functions and procedures that enables spatial data to be stored, accessed, and analyzed
  - **IBM DB2 Spatial Extender** can be used to enable any edition of DB2 with support for spatial types
  - **Microsoft SQL Server** has support for spatial types since 2012
    - geometry type for Euclidean coordinates
    - geography type for earth coordinates
  - **Rasdaman** provides support for raster images.

Exercises and Exam

- During the lectures we will solve important representative in-class examples that help you to understand the material.
- Bring a laptop with PostgreSQL client, shell, vpn, and possibly pgadmin.
- There will be an implementation exercise with an extension of PostgreSQL that supports the processing of time intervals. Requires install and compilation or vpn.

- The exam takes place January 6, 2020.
- The exam is written (about 1h) or oral (about 20min).
- It is important that you choose appropriate examples to illustrate your answers.

Schedule (tentative)

16.09	Introduction and motivation
23.09	Introduction and motivation
30.09	Time domain, granularities
07.10	Temporal semantics
14.10	Temporal semantics
21.10	Temporal data models
28.10	Temporal extensions of SQL
04.11	Temporal extensions of SQL
11.11	Temporal query processing
18.11	Temporal query processing
25.11	Spatial database systems
02.12	Spatial database systems
09.12	Spatial database systems
16.12	Summary

Literature (tentative)

- K. Kulkarni, J.-E. Michels, *Temporal features in SQL: 2011*, ACM SIGMOD Record 41.3 34-43, 2012.
- A. Dignös, M. H. Böhlen, J. Gamper, C. S. Jensen, *Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries*, ACM Trans. Database Syst. 41(4): 26:1-26:46, 2016.
- M. H. Böhlen, C. S. Jensen, Temporal Data Model and Query Language Concepts, *Encyclopedia of Information Systems*, Volume 4, Elsevier Science, 2003.
- D. Gao, C. S. Jensen, R. T. Snodgrass, and M. D. Soo: Join operations in temporal databases. *VLDB Journal*, 14:2–29, 2005.
- B. Moon, I. F. Vega Lopez, and V. Immanuel: Efficient algorithms for large-scale temporal aggregation. *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, 2004.
- R. H. Güting: An introduction to spatial database systems. *VLDB Journal* 3:357–399 (1994)
- D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao: Query processing in spatial network databases. In *Proc. of the VLDB*, 2003.

Basic Definitions

- **Mini-world**: The part of the real world we are interested in
- **Data**: Known facts about the mini-world that can be recorded
- **Database (DB)**: A collection of related data
- **Database Management System (DBMS)**: A software package to facilitate the creation/maintenance of databases
- **Database System**: DB + DBMS
- **Meta Data**: Information about the structure of the DB
  - Meta data is organized as a DB itself

The Relational Data Model/1

- Data are stored in relations/tables

employee

Name	Dept	Salary
Tom	SE	23K
Lena	DB	33K

department

Dname	Manager	Address
SE	Tom	Boston
DB	Lena	Tucson

project

ProjID	Dept	From	To
14	SE	2005-01-01	2005-12-31
173	SE	2005-04-15	2006-10-31
201	DB	2005-04-15	2006-03-31

The Relational Data Model/2

- A **domain** D is a set of atomic data values.
  - phone numbers, names, grades, birthdates, departments
  - each domain includes the special value `null` for unknown or missing
- With each domain a **data type** or format is specified.
  - 5 digit integers, yyyy-mm-dd, characters
- An **attribute** A_i describes the role of a domain in a relation schema.
  - PhoneNr, Age, DeptName
- A **relation schema** $R(A_1, ..., A_n)$ is made up of a relation name R and a list of attributes.
  - $employee(Name, Dept, Salary)$,
- A **tuple** t is an ordered list of values $t = (v_1, ..., v_n)$ with $v_i \in dom(A_i)$.
  - $t = (Tom, SE, 23K)$
- A **relation** r of the relation schema $R(A_1, ..., A_n)$ is a set of n-ary tuples.
  - $r = \{(Tom, SE, 23K), (Lene, DB, 33K)\}$
- A **database** DB is a set of relations.
  - $DB = \{r, s\}$

Properties of Relations

- A relation is a **set of tuples**, i.e.,
  - **no ordering** between tuples and
  - **no duplicates** (identical tuples) exist.
- A table (in contrast to a relation) allows duplicates.
- Attributes within tuples are **ordered**.
  - $(23K, Tom, SE)$
  - At the logical level it would be possible to have unordered tuples if the correspondence between values and attributes is maintained, e.g., $\{Salary/23K, Name/Tom, Dept/SE\}$
- Query languages:
  - Relational algebra (RA)
  - Domain relational calculus (DRC)
  - SQL

Table of Contents

- Course setup and administration
  - Lectures
  - Exercises
  - Exam
  - Literature
  - Database field
- **New requirements for database systems**
- Motivation for temporal database functionality
  - What is a temporal database
  - The need for temporal databases
  - Case study

New Requirements for Database Systems/1

- **Insurance**: In how many accidents at Rigiplatz was the left side damaged and the driver was less than 25 years? Show all available pictures and sketches of such accidents. [unstructured data]
- **Product management**: Which were the three most profitable products during the past 3 months? [entity resolution]
- **Transport/logistics**: Determine the cheapest transport routes between Bolzano and Zürich and display alternative routes on a map. [shortest path]
- **Finance**: Find all high-tech stock with a risk/profit assessment that is below the one of my current portfolio and add to each the most recent analysis reports. [external information, data integration]
- **Production**: Mark each produced chip that deviates from the provided template chip. [complex data types]

New Requirements for Database Systems/2

- Highly structured information
  - ad-hoc constructed units
    - assemble/disassemble
  - ad-hoc relations between parts
    - uses, derived from, involved in
  - versions
    - alternatives, revisions, variants, configurations
  - nonstandard attribute values
    - vectors, matrices
    - geometry, maps
    - time intervals, time series
- Unstructured information
- Semistructured information

New Requirements for Database Systems/3

Standard DB technology:

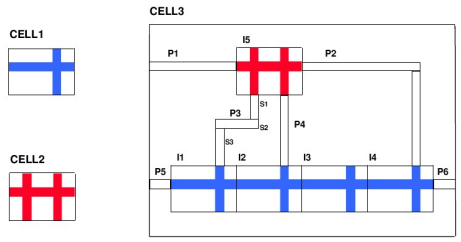
- We can do everything with a relational database systems (+ a simple programming language), but
  - Modeling becomes complex (along with subsequent SQL statements)
  - Frequent joins (to construct entities)
  - Redundancy (because of 1NF representation)
  - Unnecessary inspection of irrelevant tuples during join processing
- In many cases we can do much better if the modeling of data and query processing is improved.

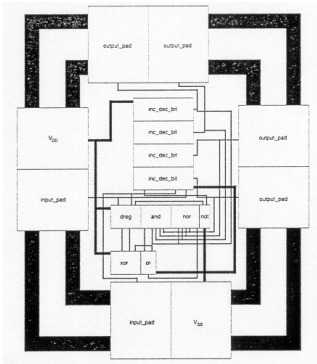
Key issue:

- Which extension make a real difference (are important enough) and should become part of DBMSs?

New Requirements for Database Systems/4

VLSI: managing VLSI designs with a database system is difficult; no good solutions for this.





New Requirements for Database Systems/5

XML: standard database systems were not prepared to deal with XML data; today they have improved a lot.

```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

New Requirements for Database Systems/6

GIS: standard database systems were not prepared to deal with geographical maps; GIS systems had to be used; today database systems are very good at handling spatial information.

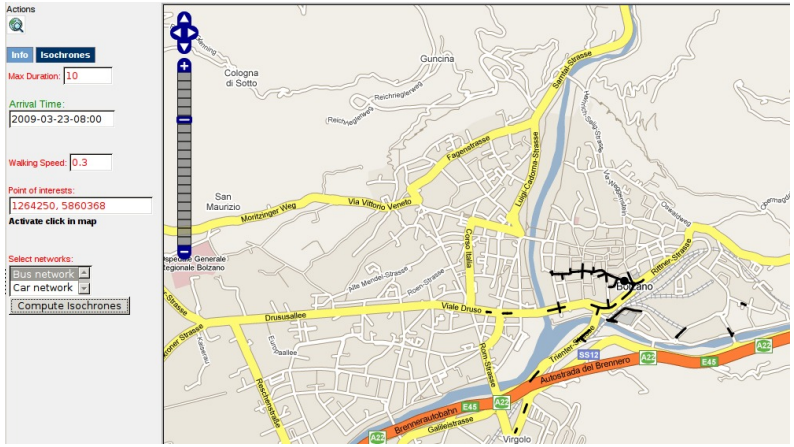


Table of Contents

- Course setup and administration
  - Lectures
  - Exercises
  - Exam
  - Literature
  - Database field
- New requirements for database systems
- **Motivation for temporal database functionality**
  - **What is a temporal database**
  - **The need for temporal databases**
  - **Case study**

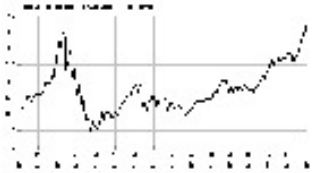
Temporal Databases

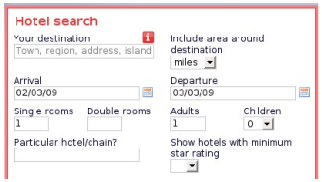
- A temporal database supports the management (= storage + querying + modification) of time-varying information.
- It is difficult to identify applications that do not work with time-varying data.

- Two kinds of temporal aspects are of general interest.
- When the data is true in reality, termed **valid time**. Example: *When was an employee in a certain department.*
- When the data is recorded in the database, termed **transaction time**. Example: *When was it recorded that the employee is in the department.*
- Existing technology provides little or no support for these.
- The main challenge addressed by temporal database systems is that of providing general-purpose built-in support for temporal concepts.

The Need for Temporal Databases/1

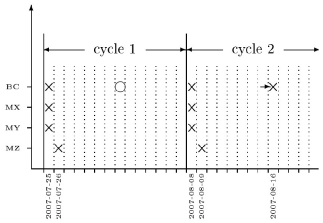
- Time is an important aspect of **all real-world phenomena**, e.g.,
  - Record-keeping applications (e.g., medical records, inventory)
  - Financial applications (e.g., banking, stock market data, trend analysis)
  - Scheduling applications (e.g., airline, train, hotel reservation)
  - Scientific applications (e.g., physics, astronomy, weather monitoring)





201 **BOZEN-TERLAN-MERAN**
BOLZANO-TERLANO-MERANO

Bozen	6.10		6.40	7.00	7.10	8.10	9.10	10.10
Krankenhaus	6.21		6.51	7.11	7.21	8.21	9.21	10.21
Drususallee	|		|	|	|	|	|	|
Siebeneich	6.24		6.54	7.14	7.26	8.26	9.26	10.26
Terlan	6.28		6.58	7.18	7.30	8.30	9.30	10.30
Vilpian	6.32		7.02	7.22	7.34	8.34	9.34	10.34
Gargazon	6.36	7.05	7.06	7.26	7.38	8.38	9.38	10.38
Burgstall	6.39	7.09	7.09	7.29	7.41	8.41	9.41	10.41
Sinich	6.44	7.13	7.14	7.34	7.46	8.46	9.46	10.46
Meran Bhf.	7.00	7.40	7.33	7.55	8.05	9.05	10.05	11.05



The Need for Temporal Databases/2

- **Limited support** for temporal data management in DBMSs
  - Conventional (non-temporal) DBs represent a snapshot of the mini-world
  - Management of temporal aspects is implemented by the application program (and not by DBMS)
  - Some time data types and functions are available in SQL, e.g., DATE, TIME, DATEADD(), DATEDIFF(), NOW()
- A **temporal database provides built-in support** for the management of temporal data/time
  - Representation of various temporal aspects, e.g., valid time, transaction time
  - Support for temporal indeterminacy, including qualitative temporal relations
  - Support for multiple calendars and granularities
  - Easy formulation of queries over time-varying data
  - Queries and modifications that operate on previous states

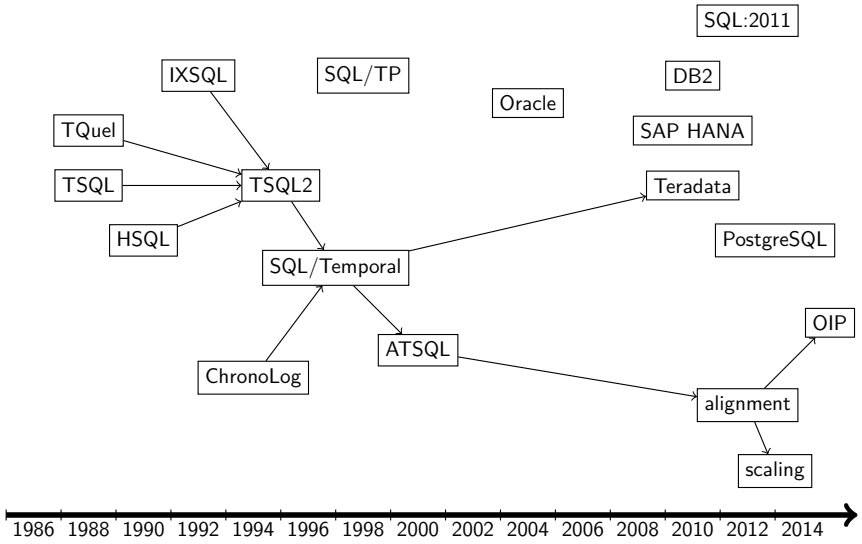
Temporal Database Research History/1

- Four overlapping phases
  - 1956–1985: **Concept development**, considering the multiple kinds of time and conceptual modeling
  - 1978–1994: **Design of query languages**
    - 1978-1990: Relational temporal query languages
    - 1990–1994: Object-oriented temporal query langauges
  - 1988–present: **Implementation aspects**, including storage structures, operator algorithms, and temporal indexes.
  - 1993–present: **Consolidation phase**
    - **Consensus glossary** of temporal database concepts http://www.cs.aau.dk/~csj/Glossary/index.html
    - Query language test suite
    - **Standardization** of key features of temporal database systems

Temporal Database Research History/2

- An **active area** for research and product developments
  - New application domains with the need for new operations
    - business intelligence, data warehousing
    - spatio-temporal and moving-object databases (e.g., mobile-phone tracking to monitor employees, company cars, and equipment)
    - data streams
  - During recent years lots of efforts from companies:
    - Oracle 10g, 2003: temporal extensions through workspace manager, time travel
    - SAP HANA, 2010: history tables
    - IBM DB2 10, 2010: Current and history tables, business (= valid) time, system (= transaction) time, time travel
    - Teradata 13.10, 2010: time travel, parts of ANSI SQL/Temporal
    - SQL:2011 standard with temporal extensions

Temporal Database Research History/3



A Case Study/1

Example: Consider a company that stores the following information about the employees: name, salary, and position.

- No temporal data
  - Schema: `Employee(Name, Salary, Job)`
  - Query: *What is Bob's salary?*

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

- Additionally store the date of birth (DoB)
  - SQL provides a data type **DATE**
  - Schema: `Employee(Name, Salary, Job, DoB` **`DATE`**`)`
  - Query: *What is Bob's date of birth?*

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

  - Finding the date of birth is analogous to determining the salary.

A Case Study/2

- Storing **history information**
  - Now the employment history shall be stored
  - Store for each tuple the time period when the tuple was/is valid
  - Schema:
    `Employee(Name, Salary, Job, Start` **`DATE`**`, End` **`DATE`**`)`

Employee

Name	Salary	Job	Start	End
Bob	60000	Assistant Provost	1995-01-01	1995-05-31
Bob	70000	Assistant Provost	1995-06-01	1995-09-30
Bob	70000	Provost	1995-10-01	1995-11-30
Bob	70000	Professor	1995-12-01	1997-12-31

- To the data model, these new columns are identical to `DoB`, but they have far-reaching consequences for queries.
- The Start and End columns model the time during which the fact is valid in the real world.

A Case Study/3

- **Temporal projection**
  - Query: *What is Bob's current salary?*

```
SELECT Salary
FROM Employee
WHERE Name = 'Bob'
AND Start <= CURRENT_DATE
AND CURRENT_DATE <= End
```

  - The query is more complicated (but still simple enough; and we agree on the result!).

A Case Study/4

- Compute the **salary history** for employees
  - Query: *What is Bob's salary history?*

Employee

Name	Salary	Job	Start	End
Bob	60000	Assistant Provost	1995-01-01	1995-05-31
Bob	70000	Assistant Provost	1995-06-01	1995-09-30
Bob	70000	Provost	1995-10-01	1995-11-30
Bob	70000	Professor	1995-12-01	1997-12-31

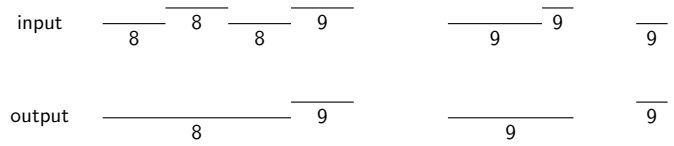
  - Intended answer:

Result relation

Name	Salary	Start	End
Bob	60000	1995-01-01	1995-05-31
Bob	70000	1995-06-01	1997-12-31

A Case Study - Coalescing/1

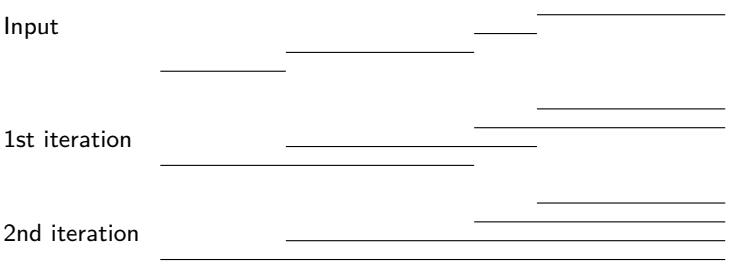
- Original query:
  What is Bob's salary history?
- Refined query:
  Longest possible periods during which Bob's salary remained constant?
- Thus, the maximal periods of time for each salary need to be determined.
- Example:



- Computing the salary history is difficult in SQL and requires the **coalescing of consecutive, value-equivalent** tuples.

A Case Study - Coalescing/2

- **Coalescing – Solution 1**: Iterative approach where pairs of tuples that are overlapping/adjacent and value-equivalent are coalesced (similar to the computation of transitive closure in graphs).
  1. Find time periods with the same salary that overlap or are adjacent.
  2. Merge these periods (either inserting new tuples or updating existing tuples).
  3. Repeat step 1 and 2 until maximal periods are constructed.
  4. Remove the non-maximal periods.



A Case Study - Coalescing/3

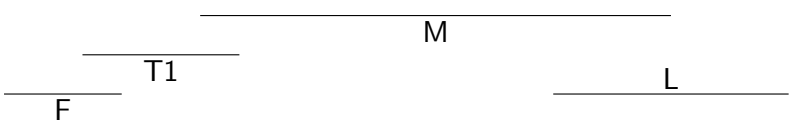
- **Coalescing – Solution 2**: SQL-1999 used common table expressions (CTEs) to formulate recursive queries.

```
WITH RECURSIVE t AS (
  SELECT * FROM r
  UNION
  SELECT ...
  FROM r, t
  WHERE ...
)
SELECT * FROM t;
```

- A recursive CTE has three parts:
  - The base query returns the initial result set.
  - The recursive query references the common table expression and its result is union-ed with the result from the previous iteration.
  - The condition in the recursive query ensures termination.

A Case Study - Coalescing/4

- **Coalescing – Solution 2**: Entirely in SQL using multiple nested **not exists** clauses
  - Search for two (possibly the same) value-equivalent tuples, `F` (first) and `L` (last)
  - Ensure that there are no holes between `F.Start` and `L.End`, i.e., all start points `M.Start` of value-equivalent tuples `M` are extended (towards `F.Start`) by another value-equivalent tuple `T1`
  - Ensure that only maximal periods result



A Case Study - Coalescing/5

- **Coalescing – Solution 3**: Using a loop and cursor
  - Use the sorting of the DBMS to fetch tuples ordered according to name, salary and start point
  - When a new tuple is fetched
    - either modify the current tuple
    - or return current tuple as a result tuple and start a new tuple
  - Only a single tuple buffer is needed
  - Below is a skeleton of a table function that can be called with `SELECT * FROM r_coal();`

```
CREATE OR REPLACE FUNCTION r_coal()
RETURNS TABLE (X INTEGER, S INTEGER, E INTEGER) AS
$$
BEGIN
...
END;
$$ LANGUAGE PLPGSQL;
```


A Case Study - Coalescing/6

- **Coalescing – Solution 4**: SQL window functions
  - Use the SQL window functions (introduced to support OLAP) to coalesce a table.
  - SQL window functions:

```
<window function> OVER (
 [PARTITION BY <expression list>]
 [ORDER BY <expression [ASC|DESC] list>]
 [ROWS|RANGE <window frame>] )
```

  - Window functions are evaluated as the very last part of an SQL statement. Below are some examples.

```
SELECT d, n, s, AVG(s) OVER (PARTITION BY d) FROM e
SELECT d, n, s, RANK() OVER (PARTITION BY d ORDER BY s) FROM e
SELECT n, s, SUM(s) OVER (ORDER BY n ROWS UNBOUNDED PRECEDING) FROM e
SELECT d, s, LAG(s) OVER (PARTITION BY d ORDER BY s) FROM e
```


A Case Study - Coalescing/7

- Window functions can be used to implement coalescing.
- The basic idea is to consider a table with all possible start and end points of the original periods.
- The cumulative counts of respectively start and end points can be used to identify the start and end of coalesced periods.

A Case Study - Coalescing/7

- **Coalescing – Solution 5**: Reorganization of the schema
  - Separate salary and position information in different relations
    - `EmpSalary(Name, Salary, Start DATE, End DATE)`
    - `EmpJob(Name, Job, Start DATE, End DATE)`

EmpSalary

Name	Salary	Start	End
Bob	60000	1995-01-01	1995-05-31
Bob	70000	1995-06-01	1997-12-31

EmpJob

Name	Job	Start	End
Bob	Ass Prov	1995-01-01	1995-09-30
Bob	Provost	1995-10-01	1995-11-30
Bob	Professor	1995-12-01	1997-12-31

A Case Study - Coalescing/8

- **Solution 5**: Reorganization of the schema
  - Computing Bob's **salary history** becomes trivial

```
SELECT Salary, Start, End
FROM EmpSalary
WHERE Name = 'Bob'
```

  - Coalescing is not needed for the salary history, but introduces the need for **temporal joins** for more complex queries.
  - The above reorganization does not ensure that there are no other queries for which coalescing is necessary.

A Case Study - Join/1

- **Temporal join** of `EmpJob` and `EmpSalary`
  - Query: *Provide the salary and position history for all employees?*

Name	Salary	Job	Start	End
Bob	60000	Ass Prov	1995-01-01	1995-05-31
Bob	70000	Ass Prov	1995-06-01	1995-09-30
Bob	70000	Provost	1995-10-01	1995-11-30
Bob	70000	Professor	1995-12-01	1997-12-31

  - Case analysis on how each tuple of *EmpSalary* overlaps each tuple of *EmpJob* is required, e.g.,
    - `EmpJob` tuple entirely contained in the `EmpSalary` tuple
    - `EmpJob` tuple overlaps the `EmpSalary` tuple

A Case Study - Join/2

- Formulation of the join query in SQL

```
SELECT E1.Name, Salary, Job, E1.Start, E1.End
FROM EmpSalary AS E1, EmpJob AS E2
WHERE E1.Name = E2.Name
AND E2.Start <= E1.Start AND E1.End <= E2.End

UNION
SELECT E1.Name, Salary, Job, E1.Start, E2.End
FROM EmpSalary AS E1, EmpJob AS E2
WHERE E1.Name = E2.Name
AND E1.Start > E2.Start
AND E2.End < E1.End AND E1.Start < E2.End

UNION
SELECT E1.Name, Salary, Job, E2.Start, E1.End
FROM EmpSalary AS E1, EmpJob AS E2
WHERE E1.Name = E2.Name
AND E2.Start > E1.Start
AND E1.End < E2.End AND E2.Start < E1.End

UNION
SELECT E1.Name, Salary, Job, E2.Start, E2.End
FROM EmpSalary AS E1, EmpJob AS E2
WHERE E1.Name = E2.Name
AND E2.Start >= E1.Start AND E2.End <= E1.End
```

A Case Study - Join/3

- Use of CASE statement

```
SELECT E1.Name, Salary, Job,
       CASE WHEN E2.Start < E1.Start
            THEN E1.Start
            ELSE E2.Start
       END,
       CASE WHEN E2.End < E1.End
            THEN E2.End
            ELSE E1.End
       END
FROM EmpSalary AS E1, EmpJob AS E2
WHERE E1.Name = E2.Name
AND E1.End >= E2.Start AND E1.Start <= E2.End
```


Summary

- Non-temporal DBMSs and their query languages provide **inadequate support for temporal aspects**.
  - Understand and illustrate the limitations of relational database systems for modeling time-varying information:
    - coalescing
    - join
    - aggregation
- Formulation of the queries must be simple.
  - *What is the average salary for each type of position*? should syntactically be (very) similar to

```
SELECT AVG(Salary), Job
FROM Employee
GROUP BY Job
```

- In a **temporal DBMS**
  - the data model more accurately reflects the reality (captures changes)
  - temporal attributes have a specific semantics
  - with support through a temporal DBMS queries shall become simpler