Temporal and Spatial Database 2014/2015 2nd semester

Introduction
SL01

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

Acknowledgements: I am indebted to Michael Böhlen for providing me the slides.

Table of Contents

This Course/1

- Course page
  - `http://www.inf.unibz.it/dis/teaching/TSDB/`
- Syllabus
  1. New requirements and motivation
  2. Time domain, timestamps, granularity, calendar
  3. Abstract and concrete temporal data models
  4. Temporal Extensions of SQL
  5. Temporal aggregation
  6. Temporal Cartesian product and join
  7. Spatial databases
  8. Query processing in spatial network databases

This Course/2

- The course is an advanced course in the area of database systems.
- I assume you have followed an introductory database course before (relational model; algebra; SQL; query processing; etc).

- The course is research based. In some cases there is no consensus about the best approach.
- The reading material consists of selected research papers or book chapters.
- Reading such research papers is important and difficult.

This Course/3

- The problems we discuss are present in many applications and are being incorporated into commercial systems.
  - Oracle: flashback, total recall
  - Teradata: current, sequenced, and non-sequenced statements
  - Microsoft Research: Immortal DB
  - PostgreSQL: PGTemporal, time travel, transaction time
  - SQL standard: PERIOD
- A number of database systems support spatial functionality
  - PostgreSQL DBMS uses the spatial extension PostGIS to implement the standardized datatype geometry and corresponding functions.
  - Oracle Spatial
  - 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 2008

Exercises

- During the lectures we will solve representative examples that help to understand the material.
- Bring a laptop with PostgreSQL client, shell, and possibly pgadmin. Or use a unibz PC.
- There will be an implementation exercise with an extension of PostgreSQL that supports the processing of time intervals.
- Solving the exercises is not mandatory but highly recommended since it is a good preparation for the exam.

Exam

- There is an oral exam at the end of the course.
- The oral exam lasts half an hour in total.
- There will be a couple of questions about selected topics from the course.
- The questions are drawn randomly.
- It is important that you illustrate the answer to your question through an appropriately chosen example.

Literature (tentative)

- C. Bettini, S. Jajodia, X. S. Wang, *Time Granularities in Databases, Data Mining, and Temporal Reasoning*, chap. 2, Springer-Verlag, July 2000.
- C. S. Jensen, M. D. Soo, and R. T. Snodgrass, Unification of Temporal Data Models, *ICDE 2003*, pp. 262-271, 1993.
- 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.

Database Publications

- **Conference Publications**
  - ACM International Conference on Management of Data (SIGMOD)
  - International Conference on Very Large Databases (VLDB)
  - IEEE International Conference on Data Engineering (ICDE)
  - International Conference on Extending Database Technology (EDBT)
- **Journal Publications**
  - ACM Transaction on Database System (TODS)
  - The VLDB Journal (VLDBJ)
  - Information Systems (IS)
  - IEEE Transactions on Knowledge and Data Engineering (TKDE)
- **DBLP Bibliography** (maintained by Michael Ley, Uni Trier)
  - `http://www.informatik.uni-trier.de/~ley/db/`
- The **dbworld** mailing list
  - `http://www.cs.wisc.edu/dbworld/`

Basic Definitions/1

- **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

Basic Definitions/2

- A DBMS provides two kind of languages
  - A **data definition language** (DDL) for specifying the database schema
    - the database schema is stored in the data dictionary
    - the content of data dictionary is called metadata
  - A **data manipulation language** (DML) for updating and querying databases, i.e.,
    - retrieval of information
    - insertion of new information
    - deletion of information
    - modification of information
- SQL, the intergalactic data speak [Stonebraker], provides DDL and DML statements.

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 value
- 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.
  - $employee(Name, Dept, Salary)$, $department(DName, Manager, Address)$

The Relational Data Model/3

- 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.
  - $DB = \{r, s\}$
  - $r = \{(Tom, SE, 23K), (Lene, DB, 33K)\}$

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**.
  - At the logical level it is possible to have unordered tuples if the correspondence between values and attributes is maintained
  - e.g., $\{Salary/23K, Name/Tom, Dept/SE\}$
  - versus $(23K, Tom, SE)$
- Query languages:
  - Relational algebra (RA)
  - Domain relational calculus (DRC), tuple relational calculus (TRC)
  - SQL

Table of Contents

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.
- **Product management**: Which were the three most profitable products during the past 3 months?
- **Transport/logistics**: Determine the cheapest transport routes between Bolzano and Zürich and display alternative routes on a map.
- **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.
- ...

New Requirements for Database Systems/2

- ...
- **Medical research**: Retrieve all X-rays images of the lung of men above 50 years who live within 5km of a toxic waste site and who have a tumor of size of at least 2cm.
- **Production**: Mark each produced chip that deviates from the provided template chip.
- **Banking**: Withdraw 150 Euro if my balance permits so and if the information of my bank card coincides with my voice and finger print.
- **Electronic commerce**: Show me all available checked shirts that are 100% cotton and include three given colors.

New Requirements for Database Systems/3

- Highly structured information
    - arbitrarily assembled units
      - assemble/disassemble
    - arbitrary relations between parts
      - uses, derived from, involved in
    - versions
      - alternatives, revisions, variants, configurations
    - nonstandard attribute values
      - vectors, matrices, geometry
    - relations between types and instances
      - generalization, specialization
- Unstructured information
- Semistructured information

New Requirements for Database Systems/4

Problems with standard database technology for nonstandard applications:

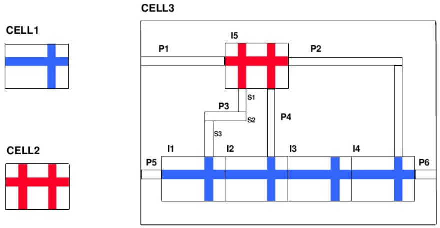
- Modeling becomes complex (and subsequent statements slow)
- Redundancy (because of 1NF representation)
- Frequent joins (to construct entities)
- Unnecessary inspection of irrelevant tuples during join processing

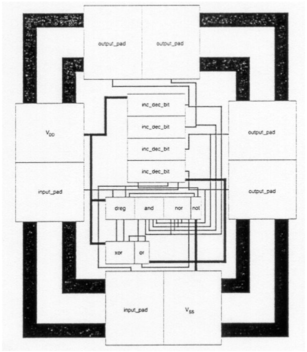
Standard database technology:

- We can do everything with a relational database systems (+ a simple programming language)
- In many cases we can do much better if the modeling of data and behavior is improved.

New Requirements for Database Systems/5

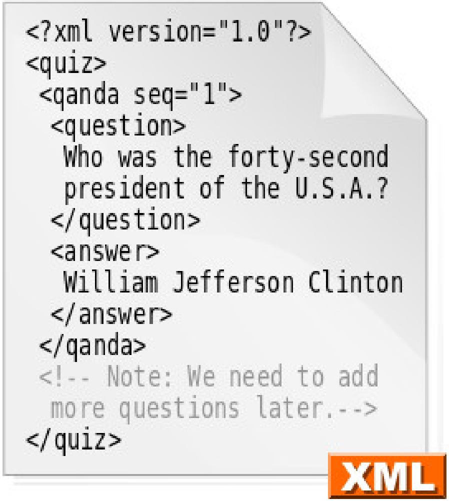
VLSI: managing VLSI designs with a database system is difficult.





New Requirements for Database Systems/6

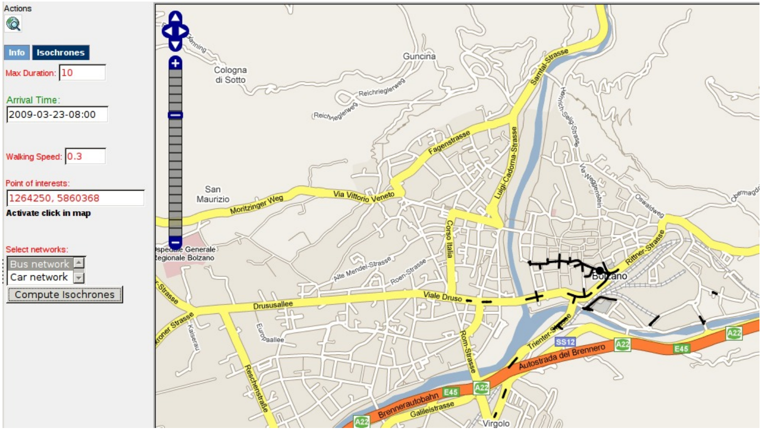
XML: standard database systems were not prepared to deal with XML data; today they improved a lot.



```
<?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>
```

New Requirements for Database Systems/7

GIS: standard database systems were not prepared to deal with geographical maps; GIS systems had to be used; today database systems improved a lot.



Actions

Info | Isochrones

Max Duration: 10

Arrival Time: 2009-03-23-08:00

Walking Speed: 0.3

Point of interests: 1264250, 5860368

Activate click in map

Select networks: Bus network, Car network

Compute Isochrones

Table of Contents

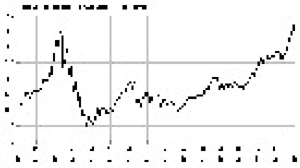
Temporal Databases

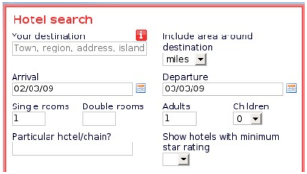
- A temporal database supports the management of time-varying information.
- It is difficult to identify applications that do not involve time-referenced data.

- 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

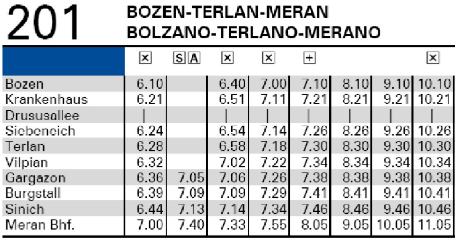
- Time is an important aspect of **all real-world phenomena**, e.g.,
  - Record-keeping applications (e.g., medical records, inventory)
  - Financial applications (e.g., banking, stock market data, trend analysis)
  - Scheduling applications (e.g., airline, train, hotel reservation)
  - Scientific applications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





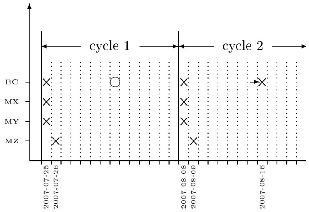
Hotel search

Your destination: Town, region, address, island	Include area around destination: miles
Arrival: 02/03/09	Departure: 03/03/09
Single rooms: 1	Double rooms:
Adults: 1	Children: 0
Particular hotel/chain?	Show hotels with minimum star rating



201 BOZEN-TERLAN-MERAN BOLZANO-TERLANO-MERANO

	☒	S A	☒	☒	⊞			☒
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 complex queries over time
  - Queries over and modification of previous states

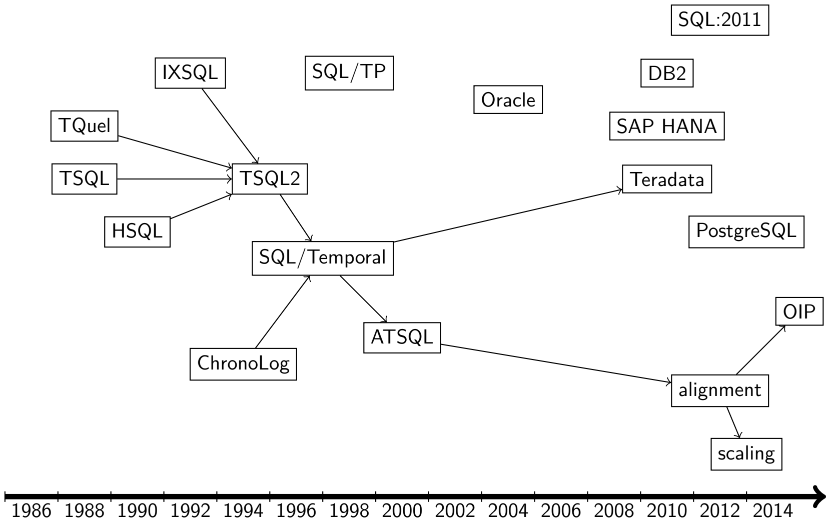
Temporal Database Research History/1

- 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

Temporal Database Research History/2

- An **active research area** today
  - Over 2000 papers produced over the past two decades
  - New application domains with the need for new operations
    - spatio-temporal and moving-object databases (e.g., mobile-phone tracking to monitor employees, company cars, and equipment)
    - data streams
    - data warehousing
  - 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 - Coalescing/1

- 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

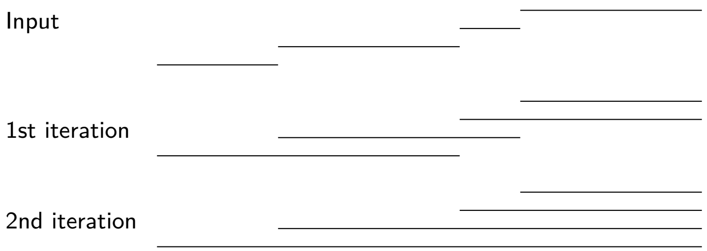
Name	Salary	Start	End
Bob	60000	1995-01-01	1995-05-31
Bob	70000	1995-06-01	1997-12-31

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

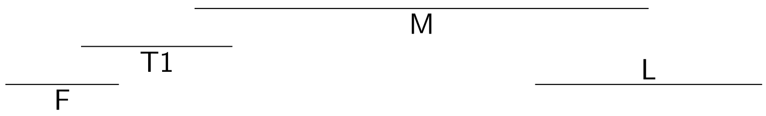
A Case Study - Coalescing/2

- **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**: 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/4

- **Coalescing – Solution 3**: Using a 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/5

- **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/6

- 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'
    ```

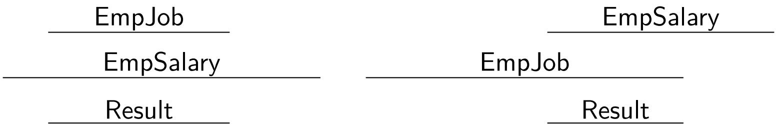
  - Coalescing is not needed for the salary history, but introduces the need for **temporal joins** for more complex queries.

A Case Study - Join/1

- **Temporal join**
  - 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
    - `EmpJob` tuple entirely contained in the `EmpSalary` tuple
    - `EmpJob` tuple overlaps the `EmpSalary` tuple
    - etc.



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

- Do the same but do not return start and end time.
- No union is required

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

A Case Study - Join/4

- 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