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

- 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
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.
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)


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/](http://www.informatik.uni-trier.de/~ley/db/)

- **The dbworld mailing list**
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
A DBMS provides two kinds of languages

- **Data Definition Language (DDL)** for specifying the database schema
  - The database schema is stored in the data dictionary
  - The content of the data dictionary is called metadata

- **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

- Data are stored in relations/tables

### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>SE</td>
<td>23K</td>
</tr>
<tr>
<td>Lena</td>
<td>DB</td>
<td>33K</td>
</tr>
</tbody>
</table>

### Department

<table>
<thead>
<tr>
<th>Dname</th>
<th>Manager</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>Tom</td>
<td>Boston</td>
</tr>
<tr>
<td>DB</td>
<td>Lena</td>
<td>Tucson</td>
</tr>
</tbody>
</table>

### Project

<table>
<thead>
<tr>
<th>ProjID</th>
<th>Dept</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>SE</td>
<td>2005-01-01</td>
<td>2005-12-31</td>
</tr>
<tr>
<td>173</td>
<td>SE</td>
<td>2005-04-15</td>
<td>2006-10-31</td>
</tr>
<tr>
<td>201</td>
<td>DB</td>
<td>2005-04-15</td>
<td>2006-03-31</td>
</tr>
</tbody>
</table>
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** \( R(A_1, \ldots, A_n) \) is made up of a relation name \( R \) and a list of attributes.
  - `employee(Name, Dept, Salary),
    department(DName, Manager, Address)`
A tuple $t$ is an ordered list of values $t = (v_1, ..., v_n)$ with $v_i \in dom(A_i)$.

$\text{t} = (\text{Tom, SE, 23K})$

A relation $r$ of the relation schema $R(A_1, ..., A_n)$ is a set of n-ary tuples.

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

A database $DB$ is a set of relations.

$DB = \{r, s\}$

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

- 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.
- **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 fingerprint.
- **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.
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
<?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.
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 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

- 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)
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
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
- **1988–present:** *Implementation aspects*, including storage structures, operator algorithms, and temporal indexes.
- **1993–present:** *Consolidation phase*
  - *Consensus glossary* of temporal database concepts
  - 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

TSQL
TQuel
IXSQL
TSQL
SQL/Temporal
SQL/TP
Oracle
DB2
SAP HANA
Teradata
PostgreSQL
SQL:2011

ChronoLog
ATSQl
alignment
OIP
scaling

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.
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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Job</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>60000</td>
<td>Assistant Provost</td>
<td>1995-01-01</td>
<td>1995-05-31</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Assistant Provost</td>
<td>1995-06-01</td>
<td>1995-09-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Provost</td>
<td>1995-10-01</td>
<td>1995-11-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Professor</td>
<td>1995-12-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>

- 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.
Temporal projection

- Query: *What is Bob’s current salary?*

```sql
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!).
Compute the salary history for employees

Query: What is Bob’s salary history?

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Job</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>60000</td>
<td>Assistant Provost</td>
<td>1995-01-01</td>
<td>1995-05-31</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Assistant Provost</td>
<td>1995-06-01</td>
<td>1995-09-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Provost</td>
<td>1995-10-01</td>
<td>1995-11-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Professor</td>
<td>1995-12-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>

Intended answer:

Result relation

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>70000</td>
<td>1995-06-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>
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.
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.

Input

1st iteration

2nd iteration
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

```
            M
     _____T1____
    ____T1_____
   _______F____
             L
```
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
    ```sql
    SELECT * FROM r_coal();
    ```

    ```sql
    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.

```sql
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
```
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.
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>70000</td>
<td>1995-06-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>

**EmpJob**

<table>
<thead>
<tr>
<th>Name</th>
<th>Job</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Ass Prov</td>
<td>1995-01-01</td>
<td>1995-09-30</td>
</tr>
<tr>
<td>Bob</td>
<td>Provost</td>
<td>1995-10-01</td>
<td>1995-11-30</td>
</tr>
<tr>
<td>Bob</td>
<td>Professor</td>
<td>1995-12-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>
Solution 5: Reorganization of the schema

- Computing Bob’s salary history becomes trivial

```sql
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?

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Job</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Ass Prov</td>
<td>1995-06-01</td>
<td>1995-09-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Provost</td>
<td>1995-10-01</td>
<td>1995-11-30</td>
</tr>
<tr>
<td>Bob</td>
<td>70000</td>
<td>Professor</td>
<td>1995-12-01</td>
<td>1997-12-31</td>
</tr>
</tbody>
</table>

- 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.

EmpJob | EmpSalary
_____ | _____
EmpSalary | EmpJob
_____ | _____
Result | Result
_____ | _____
Formulation of the join query in SQL

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

    ```sql
    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