

Part1

Review of Relational Databases and Motivation for Temporal Databases and Time Domain and Calendars

- ▶ About the course
- ▶ Review of relational databases
- ▶ The need for temporal databases
- ▶ Time domain and granularities

This Course/1

- ▶ Course page
 - ▶ <http://www.inf.unibz.it/~boehlen/NStdDBS>
- ▶ Syllabus
 1. Review DBs, motivation, time domain, granularity, calendar, now
 2. Points, intervals, temporal elements, BCDM
 3. Representational models, temporal relational algebra
 4. Temporal query languages
 5. Temporal joins, temporal aggregation
 6. Spatial databases
 7. Query processing in spatial networks

This Course/2

- ▶ The course is an advanced course and mostly targeted at MSc students.
- ▶ I assume you know basic database stuff (relational model; algebra; SQL; query processing; etc).
- ▶ The material consists of selected research papers or book chapters.
- ▶ Reading such research papers is important and difficult. (Today too many people read too many things without understanding.)
- ▶ This course is no easy since no well-established and comprehensive reference textbook exists.

This Course/3

- ▶ Each week I have planned to include a short review that gives you the possibility to apply what you have heard and assess yourself.
- ▶ We will solve and discuss the reviews here in class.
- ▶ The reviews are a good preparation for the exam.
- ▶ The exam is written. In case of a low number of participants the exam might be done orally. The tentative date for the exam is January 8, 2010.

About me/1

- ▶ My employment history:
 - ▶ 1990 - 1994 at ETH Zürich, CH
 - ▶ 1994 - 1995 at University of Arizona, USA
 - ▶ 1995 - 2003 at Aalborg University, Denmark
 - ▶ 2003 - now at Free University of Bozen-Bolzano, Italy
- ▶ In general
 - ▶ I like to be precise,
 - ▶ I like to be concrete,
 - ▶ I like examples, and
 - ▶ I like algorithms and systems.

About me/2

- ▶ Since more than fifteen years I have been an active researcher in the area of temporal database systems.
- ▶ I participated in a EU research project with researchers in temporal and spatial databases (Chorochronos).
- ▶ Since several years I have been working on research issues related to temporal, spatial, and spatio-temporal database systems.

Literature and Acknowledgements

- ▶ C. Bettini, S. Jajodia, X.S. Wang, *Time Granularities in Databases, Data Mining, and Temporal Reasoning*, chap. 2, Springer-Verlag, July 2000.
- ▶ C. Bettini, C. Dyreson, W. Evans, R. Snodgrass, S. Wang, *A Glossary of Time Granularity Concepts*, Temporal databases - Research and Practice, LNCS 1399, pp. 406-413, 1998.
- ▶ P. Ning, X. S. Wang, S. Jajodia, *An algebraic representation of calendars*, Annals of Mathematics and Artificial Intelligence, 36, pp. 5-38, 2002.
- ▶ These slides are based on slides from
 - ▶ Johann Gamper, Free University of Bozen-Bolzano, Italy
 - ▶ Christian S. Jensen, Aalborg University, Denmark
 - ▶ X. Sean Wang, University of Vermont, USA

The Field/1

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

The Field/2

The DBLP Computer Science Bibliography

maintained by [Michael Ley](#) - [Welcome](#) - [FAQ](#)

DBLP is available from several hosts: [Trier I](#) - [Trier II](#) - [ACM SIGMOD](#) - [SunSITE CE](#)

Search

- [Author](#)
- [Faceted search](#) ([L3S Research Center](#), [U. Hannover](#))
- [CompleteSearch](#) ([Holger Bast](#), [Max Planck Institut f. Inf.](#))

Bibliographies

- **Conferences:** [SIGMOD](#), [VLDB](#), [PODS](#), [ER](#), [EDBT](#), [ICDE](#), [POPL](#), ...
- **Journals:** [CACM](#), [TODS](#), [TOIS](#), [TOPLAS](#), [DKE](#), [VLDB J.](#), [Inf. Systems](#), [TPLP](#), [TCS](#), ...
- **Series:** [LNCS/LNAI](#), [IFIP](#)
- **Books:** [Reference](#) - [Collections](#) - [DB Textbooks](#)
- **By Subject:** [Database Systems](#), [Logic Prog.](#), [IR](#), ...

Full Text: [ACM SIGMOD Anthology](#)

The Field/3

- ▶ Products
 - ▶ **Oracle**
 - ▶ **DB2**
 - ▶ **Microsoft SQL Server**
 - ▶ Sybase
 - ▶ Ingres
 - ▶ Informix
 - ▶ PC database systems: Paradox, Access, FoxPro, ...
 - ▶ **PostgreSQL (open source)**
 - ▶ **MySQL (open source)**
 - ▶ MonetDB (open source)

The Field/4

- ▶ DB research and practice has many aspects
 - ▶ Design of languages
 - ▶ Development of algorithms
 - ▶ Data modeling
 - ▶ User interface design
 - ▶ Design of migration strategies
 - ▶ Distributed computing
 - ▶ High data volumes and efficiency
- ▶ New data models and applications
 - ▶ Stream data processing
 - ▶ XML/semi-structured databases
 - ▶ Temporal, spatial, moving object databases
 - ▶ Multimedia databases
 - ▶ Object oriented databases

Review of Relational Databases

- ▶ The DB field
- ▶ Basic DB definitions
- ▶ The relational data model
- ▶ Limitation of standard database systems

Basic Definitions/1

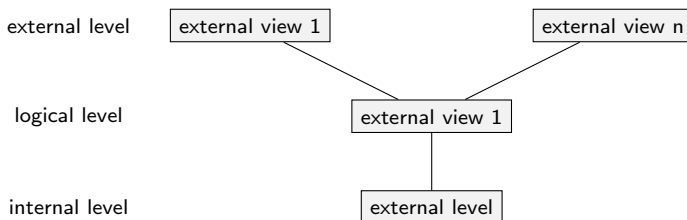
- ▶ **Mini-world:** The part of the real world we are interested in
- ▶ **Data:** Known facts about the mini-world that can be recorded
- ▶ **Database (DB):** A collection of related data
- ▶ **Database Management System (DBMS):** A software package to facilitate the creation/maintenance of databases
- ▶ **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.

ANSI/SPARC Architecture

- ▶ ANSI/SPARC three level schema architecture aims to achieve:
 - ▶ program/data independence
 - ▶ multiple user views
- ▶ Schemes are defined at three levels:
 - ▶ the **internal level** describes data storage structures and access paths; uses a physical schema
 - ▶ the **logical level** describes the structure and constraints for the whole database; uses a logical schema
 - ▶ the **external level** describes various user views for parts of the database; often uses the logical schema



DBMS Functionality

- ▶ Stores and retrieves data
- ▶ Controls redundancy
 - ▶ data, development, maintenance
- ▶ Ensures data consistency
- ▶ Enables sharing of data by multiple applications
- ▶ Supports concurrent access by multiple users
- ▶ Protects against loss of data
- ▶ Ensures security

Databases – Pros and Cons

► Pros:

- Logical
- Data abstraction
- Meta reasoning
- Self describing (data dictionary)
- Multiple user views
- Data sharing

► Cons:

- Huge and complex systems
- Restrict functionality
- Substantial overhead
- No direct data access

► When **not to use** a DBMS

- Too high costs
 - High initial investment (software, hardware, training)
 - Overhead for providing generality, security, recovery, integrity, and concurrency
- Simple, well defined, and not-changing application
- No multi-user access required
- Stringent real-time requirements

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

PId	Dept	From	To
14	SE	01.01.2005	31.12.2005
173	SE	15.04.2005	30.10.2006
201	DB	15.04.2005	31.03.2006

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** $R(A_1, \dots, A_n)$ 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 $t = (v_1, \dots, v_n)$ with $v_i \in \text{dom}(A_i)$.
 - ▶ $t = (\text{Tom}, \text{SE}, 23K)$
- ▶ A **relation** r of the relation schema $R(A_1, \dots, A_n)$ is a set of n -ary tuples.
 - ▶ $r = \{(\text{Tom}, \text{SE}, 23K), (\text{Lene}, \text{DB}, 33K)\}$
- ▶ A **database** DB is a set of relations.
 - ▶ $DB = \{r, s\}$
 - ▶ $r = \{(\text{Tom}, \text{SE}, 23K), (\text{Lene}, \text{DB}, 33K)\}$
 - ▶ $s = \{(\text{SE}, \text{Tom}, \text{Boston}), (\text{DB}, \text{Lena}, \text{Tucson})\}$

Properties of Relations

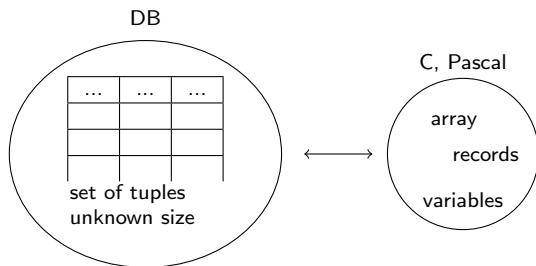
- ▶ A relation is a **set of tuples**, i.e.,
 - ▶ **no ordering** between tuples and
 - ▶ **no duplicates** (identical tuples) exist.
- ▶ A table (in contrast to a relation) allows duplicates.
- ▶ Attributes within tuples are **ordered**.
 - ▶ At the logical level it is possible to have unordered tuples if the correspondence between values and attributes is maintained
 - ▶ e.g., $\{Salary/23K, Name/Tom, Dept/SE\}$
 - ▶ versus $(23K, Tom, SE)$

Accessing DBs

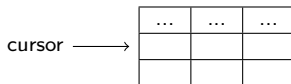
- ▶ The success of DBs also depends on the ease of data access.
- ▶ When accessing a (relational) DB two factors must be taken into account.
 - ▶ The impedance mismatch.
 - ▶ The interface to the DB.

Impedence Mismatch and Cursor

- ▶ **Impedence mismatch:** The difference between the data models (e.g., sets vs. records)



- ▶ **Cursor:** The most versatile way to connect a DB to a host language.
 - ▶ Cursors are used to resolve the impedance mismatch.
 - ▶ A cursor runs through the tuples of a relation/table.



DB Interfaces

- ▶ Various interfaces to DBs exist, e.g.,
 - ▶ Terminal interface (sqlplus, etc.)
 - ▶ OCI (Oracle Call Interface)
 - ▶ X/Open SQL CLI (Call Level Interface)
 - ▶ Embedded SQL
 - ▶ ODBC (Open Data Base Connection), iODBC for Unix
 - ▶ JDBC (Java Database Connectivity)
 - ▶ DBI (Perl DB Interface)

Oracle's OCI - Hello World/1

- ▶ The OCI is a set of C procedures to access an Oracle database:
 - ▶ orlon
 - ▶ oopen
 - ▶ oparse
 - ▶ odefin
 - ▶ obndrn
 - ▶ odescr
 - ▶ oexec
 - ▶ ofetch
 - ▶ ologf
 - ▶ oclose

Oracle's OCI - Hello World/2

```
/*
export ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
export ORACLE_SID=XE
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH;

gcc -c -I$ORACLE_HOME/rdbms/public helloworld-oci.c
gcc -o helloworld-oci helloworld-oci.o -L$ORACLE_HOME/lib -lclntsh
./helloworld-oci
*/

#include <stdio.h>
#include <stdlib.h>

#include <oratypes.h>
#include <ocidfn.h>
#include <ocikpr.h>
#include <ocidem.h>

Lda_Def lda;
ub4 hda[HDA_SIZE / 4];
Cda_Def cda;

char name[30];
char type[30];
```

Oracle's OCI - Hello World/3

```
int main() {  
  
    orlon(&lدا, hدا, "scott", -1, "tiger", -1, 0);  
  
    oopen(&cدا, &lدا, (text *) 0, -1, -1, (text *) 0, -1);  
  
    oparse(&cدا, "SELECT * FROM cat", -1, FALSE, 2);  
  
    odefin(&cدا,1,&name,30,STRING_TYPE,-1,0,0,-1,-1,0,0);  
    odefin(&cدا,2,&type,30,STRING_TYPE,-1,0,0,-1,-1,0,0);  
  
    oexec(&cدا);  
  
    for (;;) {  
        if (ofetch(&cدا)) break;  
        printf(" '%s' '%s'", name, type);  
    }  
  
    oclose(&cدا);  
    ologof(&lدا);  
    exit(0);  
}
```

ODBC - Hello World/1

- ▶ ODBC is a set of C procedures to access any SQL database, e.g.,
 - ▶ SQLAllocEnv
 - ▶ SQLAllocConnect
 - ▶ SQLConnect
 - ▶ SQLDescribeCol
 - ▶ SQLAllocStmt
 - ▶ SQLPrepare
 - ▶ SQLBindCol
 - ▶ SQLSetParam
 - ▶ SQLFetch
 - ▶ SQLDisconnect
 - ▶ SQLExecute
 - ▶ SQLFreeConnect
 - ▶ SQLExecDirect
 - ▶ SQLFreeEnv
 - ▶ SQLFreeStmt
- ▶ ODBC standardizes meta data access (listing of tables, columns, etc).

ODBC - Hello World/2

```
/*
    export ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
    export ORACLE_SID=XE
    export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH;

    gcc -c -I$ORACLE_HOME/rdbms/public/ helloworld-odbc.c
    gcc -o helloworld-odbc helloworld-odbc.o -lodbc
    ./helloworld-odbc
*/

#include <stdio.h>

#include <sql.h>
#include <sqlext.h>

SQLHENV hEnv;
SQLHDBC hDbc;
SQLHSTMT hstmt;

char name[30];
char type[30];
int rc;
```

ODBC - Hello World/3

```
int main() {

    rc=SQLAllocEnv(&hEnv);
    rc=SQLAllocConnect(hEnv,&hDbc);
    rc=SQLSetConnectOption(hDbc,SQL_AUTOCOMMIT,SQL_AUTOCOMMIT_OFF);
    rc=SQLConnect(hDbc, "OracleODBC-10g", SQL_NTS, "scott", SQL_NTS, "tiger", SQL_NTS);

    rc=SQLAllocStmt(hDbc,&hstmt);
    rc=SQLPrepare(hstmt, "select * from cat", SQL_NTS);

    rc=SQLExecute(hstmt);

    rc=SQLBindCol(hstmt,1,SQL_C_CHAR,(SQLPOINTER)name,30,NULL);
    rc=SQLBindCol(hstmt,2,SQL_C_CHAR,(SQLPOINTER)type,30,NULL);

    for (;;) {
        if (SQLFetch(hstmt)) break;
        printf(" '%s' '%s'", name, type);
    }
    rc=SQLFreeStmt( hstmt, SQL_DROP );

    rc=SQLDisconnect(hDbc);
    rc=SQLFreeConnect(hDbc);
    rc=SQLFreeEnv(hEnv);

}
```

ODBC Setup

File \$HOME/.odbc.ini (data source configuration file)

```
[OracleODBC-10g]
Driver = Oracle 10g ODBC driver
DSN = OracleODBC-10g
FailoverDelay = 10
FailoverRetryCount = 10
FetchBufferSize = 64000
```

File /etc/odbcinst.ini (all installed drivers on your computer)

```
[Oracle 10g ODBC driver]
Description = Oracle ODBC driver for Oracle 10g
Driver = /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/lib/libsqora.so.10.1
Setup =
FileUsage =
CPOutput =
CPReuse =
```

JDBC - Hello World/1

- ▶ JDBC is a set of Java procedures to access any SQL database, e.g.,
 - ▶ `getConnection`
 - ▶ `createStatement`
 - ▶ `executeQuery`
 - ▶ `getColumnName`
 - ▶ `getColumnType`
 - ▶ `getString`
 - ▶ `getResultSet`
 - ▶ `getObject`
 - ▶ `executeUpdate`
 - ▶ `execute`
 - ▶ `close`
- ▶ JDBC standardizes meta data access (listing of tables, columns, etc).

JDBC - Hello World/2

```
/*
export ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
export ORACLE_SID=XE
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH;

javac helloworld-jdbc.java
java -classpath $ORACLE_HOME/jdbc/lib/ojdbc14.jar:. HelloWorldJDBC
*/

import java.sql.*;
```

JDBC - Hello World/3

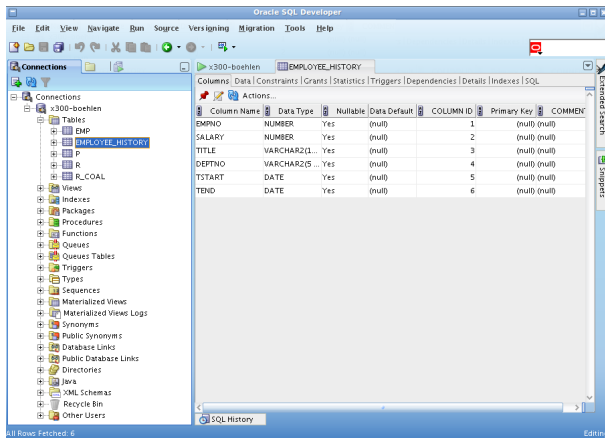
```
class HelloWorldJDBC {  
  
    public static void main (String args [])  
        throws SQLException, ClassNotFoundException  
    {  
  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
  
        Connection conn =  
            DriverManager.getConnection("jdbc:oracle:oci:@", "scott", "tiger");  
  
        Statement stmt = conn.createStatement();  
  
        ResultSet rset = stmt.executeQuery("select * from cat");  
  
        while (rset.next ())  
            System.out.println (rset.getString(1) + " " + rset.getString(2) );  
        }  
    }  
}
```

Embedded SQL

- ▶ Extended versions of, e.g., C, Pascal, and Fortran allow to embed SQL statements.
- ▶ A precompiler compiles these languages to, e.g., C with OCI library calls.
- ▶ The idea is that C with embedded SQL is easier to use than C with OCI calls.
- ▶ Embedded SQL is standardized (ISO, ANSI).

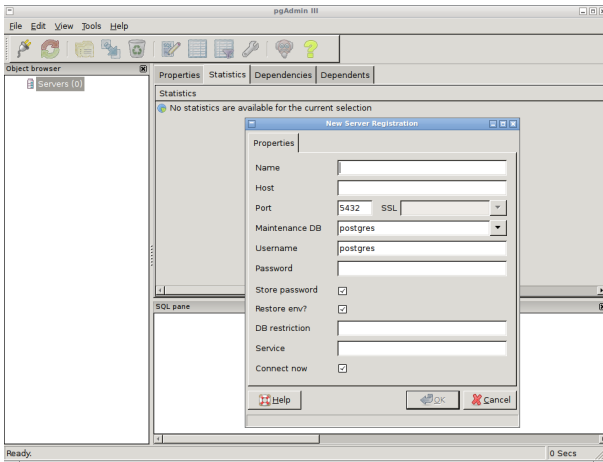
Oracle SQL Developer

- ▶ Oracle SQL Developer is a graphical tool for database development.
- ▶ With SQL Developer you can browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements.



pgAdmin

- ▶ pgAdmin is the administration and development platform for PostgreSQL.
- ▶ The graphical interface supports all PostgreSQL features, from writing simple SQL queries to developing complex databases.



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.

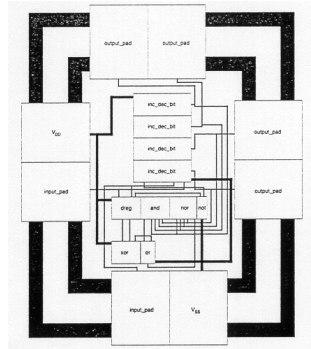
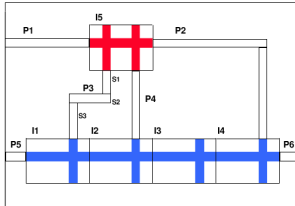
CELL1



CELL2

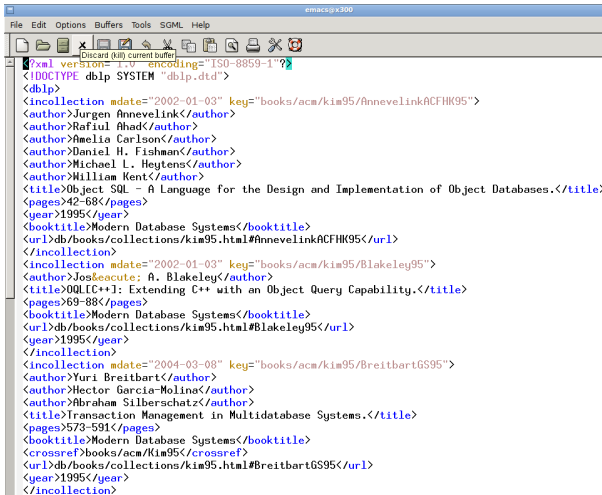


CELL3



New Requirements for Database Systems/6

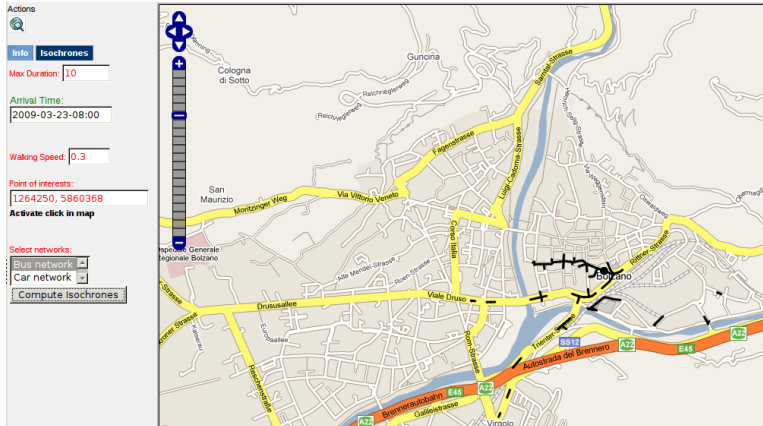
XML: standard database systems were not prepared to deal with XML data; today they improved a lot.

A screenshot of an Emacs editor window titled 'emacs@x300'. The window shows an XML document with a root element 'dblp' and several 'incollection' entries. The XML is color-coded: tags are blue, text is black, and some attributes like 'mdate' and 'key' are in red. The first 'incollection' entry is for 'Object SQL - A Language for the Design and Implementation of Object Databases' by Jürge Anneveinlink and others, published in 1995. The second entry is for 'Modern Database Systems' by Jos A. Blakeley, also published in 1995. The third entry is for 'Transaction Management in Multidatabase Systems' by Yuri Breitbart and others, published in 1995. The XML includes attributes for 'mdate', 'key', 'pages', 'booktitle', 'crossref', and 'url'.

```
?xml version="1.0" encoding="ISO-8859-1"?
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
  <incollection mdate="2002-01-03" key="books/acm/kim95/AnneveinlinkACFHK95">
    <author>Jürge Anneveinlink</author>
    <author>Rafiul Ahad</author>
    <author>Amelia Carlson</author>
    <author>Daniel H. Fishman</author>
    <author>Michael L. Heytens</author>
    <author>William Kent</author>
    <title>Object SQL - A Language for the Design and Implementation of Object Databases.</title>
    <pages>42-68</pages>
    <year>1995</year>
    <booktitle>Modern Database Systems</booktitle>
    <url>db/books/collections/kim95.html#AnneveinlinkACFHK95</url>
  </incollection>
  <incollection mdate="2002-01-03" key="books/acm/kim95/Blakeley95">
    <author>Jos A. Blakeley</author>
    <title>SQL[C++]: Extending C++ with an Object Query Capability.</title>
    <pages>69-88</pages>
    <booktitle>Modern Database Systems</booktitle>
    <url>db/books/collections/kim95.html#Blakeley95</url>
    <year>1995</year>
  </incollection>
  <incollection mdate="2004-03-08" key="books/acm/kim95/BreitbartGS95">
    <author>Yuri Breitbart</author>
    <author>Hector Garcia-Molina</author>
    <author>Abraham Silberschatz</author>
    <title>Transaction Management in Multidatabase Systems.</title>
    <pages>573-591</pages>
    <booktitle>Modern Database Systems</booktitle>
    <crossref>books/acm/Kim95</crossref>
    <url>db/books/collections/kim95.html#BreitbartGS95</url>
    <year>1995</year>
  </incollection>
```

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.



Motivation for Temporal Database Functionality

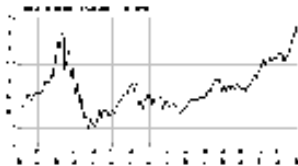
- ▶ What is a temporal database
- ▶ The need for temporal databases
- ▶ A case study

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 built-in support in all or most aspects of data management for primarily these 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 management)
 - ▶ Financial applications (e.g., banking, stock market data, trend analysis)
 - ▶ Scheduling applications (e.g., airline, train, hotel reservation)
 - ▶ Scientific applications (e.g., astronomy, physics, weather monitoring)



Hotel search

Your destination

☐ Include area around destination
miles

Arrival

Departure

Single rooms Double rooms

Adults Children

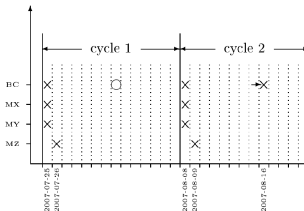
Particular hotel/chain?

☐ Show hotels with minimum star rating

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 commercial 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

- ▶ 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 languages
 - ▶ 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
 - ▶ TSQL2
- ▶ Still 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

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, Position)
 - ▶ Query: *What is Bob's salary?*

```
SELECT Salary  
FROM Employee  
WHERE Name = 'Bob'
```
- ▶ Additionally store the date of birth
 - ▶ SQL provides a data type **DATE**
 - ▶ Schema: Employee(Name, Salary, Position, DateofBirth DATE)
 - ▶ Query: *What is Bob's date of birth?*

```
SELECT DateofBirth  
FROM Employee  
WHERE Name = 'Bob'
```
 - ▶ Finding the employee's 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, Position, Start DATE, End DATE)

Employee

Name	Salary	Position	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 DateofBirth, but they have far-reaching consequences for queries.

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).

A Case Study - Coalescing/1

- Compute the **salary history** for employees

- Query: *What is Bob's salary history?*

Employee

Name	Salary	Position	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

- Exact 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 very difficult in SQL and requires the **coalescing of consecutive, value-equivalent** tuples.

A Case Study - Coalescing/2

- **Coalescing – Solution 1:** Iterative approach with SQL embedded into a loop, where pairs of overlapping/adjacent, value-equivalent tuples are coalesced (similar to the computation of transitive closure in graphs).
 1. Find those time periods with the same salary that overlap or are adjacent.
 2. Merge these periods by extending the earlier period.
 3. Repeat step 1 and 2 until maximal periods are constructed.
 4. Remove the non-maximal periods.

Input



1st iteration



2nd iteration



A Case Study - Coalescing/3

```
CREATE TABLE T(Salary,Start,End) AS
  SELECT Salary, Start, End FROM Employee WHERE Name = 'Bob'

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

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

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



- ▶ FO formula to express the basic idea to compute a coalesced tuple z ($f, l, m, t1, t2$ denote tuples)

$$z(X, [S, E]) \Leftrightarrow$$

$$r(X, [S, -]) \wedge r(X, [-, E]) \wedge S < E \wedge$$

$$\forall A(r(X, [A, -]) \wedge S < A < E \Rightarrow \exists U, V(r(X, [U, V]) \wedge U < A \leq V)) \wedge$$

$$\neg \exists A, B(r(X, [A, B]) \wedge (A < S \leq B \vee A \leq E < B))$$

A Case Study - Coalescing/5

```
CREATE TABLE T(Salary,Start,End) AS
  SELECT Salary, Start, End FROM Employee WHERE Name = 'Bob'

SELECT DISTINCT F.Salary, F.Start, L.End
FROM T AS F, T AS L
WHERE F.Salary = L.Salary
  AND F.Start < L.End
  AND NOT EXISTS (
    SELECT *
    FROM T AS M
    WHERE M.Salary = F.Salary
      AND F.Start < M.Start AND M.Start < L.End
      AND NOT EXISTS (
        SELECT *
        FROM T AS T1
        WHERE T1.Salary = F.Salary
          AND T1.Start < M.Start AND M.Start <= T1.End))
  AND NOT EXISTS (
    SELECT *
    FROM T AS T2
    WHERE T2.Salary = F.Salary
      AND ((T2.Start < F.Start AND F.Start <= T2.End)
        OR (T2.Start < L.End and L.End < T2.End)))
```

A Case Study - Coalescing/6

► Coalescing – Solution 3: Using a cursor

- Scan the relation and create a sorted list of periods in memory
- Scan the list and produce the coalesced tuples

```
DECLARE emp_cursor CURSOR FOR  
SELECT Salary, Start, End FROM Employee WHERE Name = 'Bob';
```

```
OPEN emp_cursor;
```

```
while more tuples
```

```
    FETCH emp_cursor INTO :salary, :start, :stop;  
    find position in linked list that is sorted by salary and  
    start date; insert Salary, Start, End;
```

```
CLOSE emp_cursor;
```

```
iterate through the linked list, printing out salaries along  
with coalesced time periods
```

A Case Study - Coalescing/7

► Coalescing – Solution 4: Using a cursor

- Use the sorting of the DBMS to fetch tuples ordered according to salary and start point
- When a new tuple is fetched modify the current tuple or return current tuple as a result tuple and start a new tuple
- Only a single tuple buffer is needed

```
/*  
    gcc -c -I$ORACLE_HOME/rdbms/public/ coal.c  
    gcc -o coal coal.o -lodbc  
    ./coal  
*/  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <sql.h>  
#include <sqlext.h>  
  
SQLHENV he;  
SQLHDBC hc;  
SQLHSTMT hs1, hs2;  
int cS, cE, cX, nS, nE, nX;
```

A Case Study - Coalescing/8

Preparation of statements and parameters:

```
int main() {  
  
    SQLAllocEnv(&he);  
    SQLAllocConnect(he, &hc);  
    SQLConnect(hc,"OracleODBC-10g",SQL_NTS,"scott",SQL_NTS,"tiger",SQL_NTS);  
    SQLAllocHandle(SQL_HANDLE_STMT, hc, &hs1);  
    SQLAllocHandle(SQL_HANDLE_STMT, hc, &hs2);  
  
    SQLExecDirect(hs1, "SELECT S, E, X FROM r ORDER BY X, S", SQL_NTS);  
    SQLBindCol(hs1, 1, SQL_INTEGER, &nS, sizeof(int), NULL);  
    SQLBindCol(hs1, 2, SQL_INTEGER, &nE, sizeof(int), NULL);  
    SQLBindCol(hs1, 3, SQL_INTEGER, &nX, sizeof(int), NULL);  
  
    SQLFetch(hs1);  
    cS = nS; cE = nE; cX = nX;  
  
    SQLPrepare(hs2, "INSERT INTO r_coal VALUES (?, ?, ?)", SQL_NTS);  
    SQLBindParameter(hs2,1,SQL_PARAM_INPUT, SQL_INTEGER, SQL_INTEGER,  
        sizeof(int), 0, &cX, 0, 0);  
    SQLBindParameter(hs2,2,SQL_PARAM_INPUT, ..., &cS, 0, 0);  
    SQLBindParameter(hs2,3,SQL_PARAM_INPUT, ..., &cE, 0, 0);  
}
```

A Case Study - Coalescing/9

Processing the data tuple by tuple and incremental construction of coalesced information.

```
while (SQLFetch(hs1) != SQL_NO_DATA) {  
    if (cX == nX && nS <= cE) {  
        if (nE > cE) cE = nE;  
    } else {  
        SQLExecute(hs2);  
        cS = nS; cE = nE; cX = nX;  
    }  
}  
SQLExecute(hs2);  
  
SQLDisconnect(hc);  
SQLFreeConnect(hc);  
SQLFreeEnv(he);  
}
```

A Case Study - Coalescing/10

► **Solution 5:** Reorganization of the schema

- Separate salary and position information in different relations
 - Employee1(Name, Salary, Start DATE, End DATE)
 - Employee2(Name, Position, Start DATE, End DATE)

Employee1

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

Employee2

Name	Position	Start	End
Bob	Assistant Provost	1995-01-01	1995-09-30
Bob	Provost	1995-10-01	1995-11-30
Bob	Professor	1995-12-01	1997-12-31

- Computing Bob's **salary history** becomes trivial

```
SELECT Salary, Start, End
FROM Employee1
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	Position	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

- Case analysis on how each tuple of *Employee1* overlaps each tuple of *Employee2* is required
 - *Employee2* tuple entirely contains the *Employee1* tuple
 - *Employee2* tuple overlaps the *Employee1* tuple
 - etc.



A Case Study - Join/2

► Formulation of the join query in SQL

```
SELECT E1.Name, Salary, Position, E1.Start, E1.End
FROM Employee1 AS E1, Employee2 AS E2
WHERE E1.Name = E2.Name
      AND E2.Start <= E1.Start AND E1.End <= E2.End
UNION
SELECT E1.Name, Salary, Position, E1.Start, E2.End
FROM Employee1 AS E1, Employee2 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, Position, E2.Start, E1.End
FROM Employee1 AS E1, Employee2 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, Position, E2.Start, E2.End
FROM Employee1 AS E1, Employee2 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, Position
FROM Employee1 AS E1, Employee2 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, Position,  
       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 Employee1 AS E1, Employee2 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**.
- ▶ Formulation of the queries must be simple.
 - ▶ *What is Bob's salary history?* should syntactically be (very) similar to

```
SELECT Salary
FROM Employee
WHERE Name = 'Bob'
```
 - ▶ *Provide the salary and position history for all employees?* should syntactically be (very) similar to

```
SELECT Employee1.Name, Salary, Position
FROM Employee1, Employee2
WHERE Employee1.Name = Employee2.Name
```
- ▶ In a **Temporal DBMS**
 - ▶ the data model more accurately reflects the reality
 - ▶ temporal attributes have a specific semantics
 - ▶ queries become simpler

September 18, Summary and Refresher/1

- ▶ The relational model: beautifully simple
 - ▶ domain, data type, attribute, schema, tuple, relation, database
- ▶ How to access databases (hello world examples):
 - ▶ OCI (Oracle call interface, proprietary, Oracle)
 - ▶ ODBC (Open database connectivity, Microsoft standard, widely supported)
 - ▶ JDBC (Java standard, widely supported)
- ▶ We use cursors to handle the impedance mismatch
- ▶ Adding time attributes to tables is simple but does not provide good support to answer certain types of temporal requests (coalescing, aggregation, etc).
- ▶ Doing a join gets more complex than in the nontemporal case. The different position of intervals with respect to each other must be considered. It is not difficult but leads to surprisingly complex code.

September 18, Summary and Refresher/2

- ▶ Coalescing is an example of a simple request that is very difficult to answer using SQL. Possible solutions are:
 - ▶ repeatedly find and merge pairs of overlapping tuples (using a self join) until no more merging is possible
 - ▶ declarative SQL-98 statement with subqueries
 - ▶ Idea: find first and last tuple of a chain by checking that between first and last there are no holes and first and last can not be extended.
 - ▶ Extract the data from the database and implement the coalescing in Java, C, ... by constructing a sorted list without adjacent or overlapping tuples.
 - ▶ Extract the data ordered by explicit attribute and start point (the DBMS is very good at sorting) and implement the coalescing in Java, C, ... Only a buffer for one tuple is needed.
 - ▶ Reorganize the schema (avoid coalescing as much as possible; sometimes coalescing might still be necessary since we are not aware of all possible queries and because of updates; leads to more joins).

Time Domain and Calendars

- ▶ Time domain
- ▶ Time granularity
- ▶ Calendars
- ▶ Modeling instants, periods, and intervals
- ▶ Now

Time Domain/1

► Time domain/ontology

- Specifies the basic building blocks of time
- Time is generally modeled as an arbitrary **set of instants/points** with an imposed total order, e.g., $(\mathbb{N}, \leq)$
- Additional axioms introduce more refined models of time

► Structure of time

► Linear time

- total order
- Time advances from past to future in a step-by-step fashion

► Branching time (possible future or hypothetical model)

- Time is linear from the past to now, where it then divides into several time lines
- Along any future path, additional branches may exist
- Structure is a tree rooted at now
- partial order

Time Domain/2

- ▶ **Density** of time
 - ▶ **Discrete time**
 - ▶ Time instants are non-decomposable units of time with a positive duration, called **chronons**
 - ▶ Chronon is the smallest duration of time that can be represented
 - ▶ Isomorphic to natural numbers
 - ▶ **Dense time**
 - ▶ Between any two instants of time another instant exists
 - ▶ Isomorphic to rational numbers
 - ▶ **Continuous time**
 - ▶ Dense and no “gaps” between consecutive instants
 - ▶ Time instants are durationless
 - ▶ Isomorphic to the real numbers
- ▶ **Boundness** of time
 - ▶ Time can be bounded in the past and/or in the future, i.e., first and/or last time instant exists
- ▶ **Relative** (unanchored) versus **absolute** (anchored) time
 - ▶ “9 AM, January 1, 1996” is an absolute time
 - ▶ “9 hours” is a relative time

Time Domain/3

- ▶ Humans perceive time as continuous
- ▶ A **discrete linear time model** is generally used in temporal databases for several practical reasons:
 - ▶ Measures of time are generally reported in terms of chronons
 - ▶ Natural language references are compatible with chronons, e.g., 4:30 pm means over some period/chronon around this time
 - ▶ Chronons allow easily to model durative events
 - ▶ Any implementation needs a discrete encoding of time
- ▶ Problem to represent **continuous movement** in a discrete model
 - ▶ An object is continuously moving from point A to point B

Granularities and Calendars

- ▶ The goal when defining a calendar is to not enumerate all time points but to have a compact definition of real world calendars.
- ▶ A compact definition can be used as a starting point for compact representations, efficient implementations, etc.
- ▶ We give an algebraic definition of a calendar in terms of natural granularities.

Time Granularity/1

- ▶ **Granularity:** Intuitively, a discrete unit of measure for a temporal datum that supports a user-friendly representation of time, e.g.,
 - ▶ birthdates are typically measured at granularity of days
 - ▶ business appointments to granularity of hours
 - ▶ train schedules to granularity of minutes
- ▶ Mixed granularities are of basic importance to modeling real-world temporal data
- ▶ Mixing granularities create problems
 - ▶ What are the semantics of operations with operands at differing granularities?
 - ▶ How to convert from one granularity to another?
 - ▶ How expensive is maintaining and querying times at different granularities?

Time Granularity/2

- **Example:** Airline flight database

FlightDepartures

Flight#	Time
53	'1994-11-20 14:38'
200	'1994-11-25 14:34'
653	'1994-11-27 12:38'
658	'1994-11-30 10:03'

Vacations

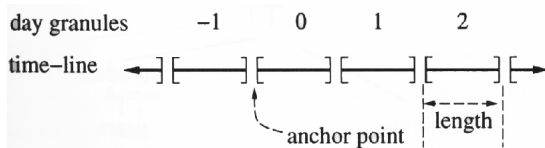
Vacation	Time
Labor Day	'[1994-09-01, 1994-09-03]'
Thanksgiving	'[1994-11-24, 1994-11-28]'
Christmas	'[1994-12-24, 1994-12-26]'

- Data are stored at **different granularities**
 - Flight departures are recorded at granularity of minutes
 - Vacations are stored at granularity of days, each tuple storing a period of days
- Query: *Which flights leave during the Thanksgiving vacation?*

```
SELECT *  
FROM Vacations, FlightDepartures  
WHERE Vacation = 'Thanksgiving'  
      AND Vacations.Time OVERLAPS FlightDepartures.Time
```
- Problems
 - Query processor needs to know the relationships between minutes and days!
 - Why is *overlaps* evaluated at granularity of days rather than minutes?

Time Granularity/3

- ▶ **Granularity:** More formally, a partitioning of the time line (chronons) into a finite set of segments, called **granules**.
- ▶ The **partitioning scheme** of a granularity is specified by
 - ▶ the **length** (or size) of each granule and
 - ▶ an **anchor point**, where the partitioning begins



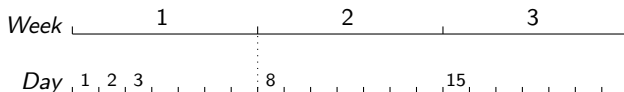
- ▶ The timeline is partitioned into granules, each the size of the partitioning length, beginning from the anchor point, and extending forwards and backwards
- ▶ The granules are labeled with their distance from the anchor point. Labels do not have to be contiguous. A granularity maps a label to the corresponding set of chronons.
- ▶ Assume granularity *Week*. Let the chronons be *Day*. Then the granule “week 2” represents the chronons {8, 9, 10, 11, 12, 13, 14}, i.e., $\text{Week}(2) = \{8, 9, 10, 11, 12, 13, 14\}$.

► Properties of a granularity

- A granularity creates a **discrete image**, in terms of granules, of a (possibly continuous) time-line.
- The smallest possible granularity is that of a **chronon**, the largest is the **entire time-line**.
- Within a given granularity, the set of granules is **well-ordered**
 - *beginning* and *forever* are the least and greatest values, respectively
- The partitioning can be **complete** (e.g., weeks, month) or **incomplete** (e.g., business weeks, holidays)
- The length of the granules can be **fixed** or **variable**
 - In reality, partitioning by using a single, fixed length is impractical, and most common granularities divide the time-line into partitions of differing length
 - A year has 365 or 366 days
 - A month varies between 28, 29, 30, and 31 days

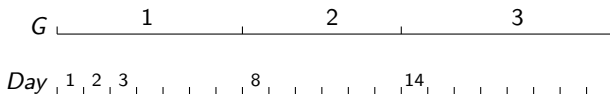
Granularity Operations/1

- ▶ $\text{Group}(G, \text{StartIndex}, \text{NumGrans})$
- ▶ Start at granule StartIndex and repeatedly group NumGrans granules into one granule
- ▶ $\text{Week} = \text{Group}(\text{Day}, 1, 7)$



Granularity Operations/2

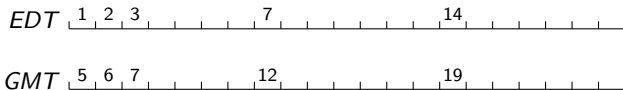
- ▶ $\text{Alter}(G2, G1, l, k, m)$
- ▶ Intuition: periodically expand/shrink granules of $G1$ in terms of granules of $G2$.
- ▶ Partition $G1$ into groups of m granules; each l -th granule of $G1$ has k extra/fewer ticks.
- ▶ $G = \text{Alter}(\text{Day}, \text{Week}, 2, -1, 3)$: in the 2nd week shrink out 1 day and do this for every three weeks



- ▶ Example: add a day to February for leap years
- ▶ Leap second: $\text{Alter}(\text{Second}, \text{Day}, i, 1, \infty)$

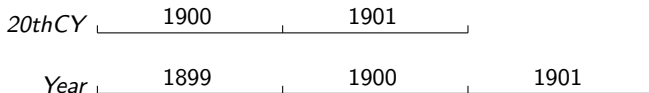
Granularity Operations/3

- ▶ $\text{Shift}(G, m)$
- ▶ Shifting operation allows to shift the index set G by m positions
- ▶ $\text{EDT} = \text{Shift}(\text{GMT}, 5)$



Granularity Operations/4

- ▶ $\text{Subset}(G, m, n)$
- ▶ Takes all the granules of G whose labels are in the interval from m to n
- ▶ $20\text{thCenturyYears} = \text{Subset}(\text{Year}, 1900, 1999)$



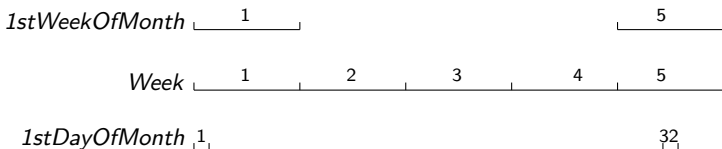
Granularity Operations/5

- ▶ $\text{Select-down}(G1, G2, k, l)$
- ▶ Selects granules of $G1$ by picking up l granules starting from the k th one in each set of granules of $G1$ contained in one granule of $G2$
- ▶ $\text{Sunday} = \text{Select-down}(\text{Day}, \text{Week}, 7, 1)$



Granularity Operations/6

- ▶ $\text{Select-up}(G1, G2, k, l)$
- ▶ Selects the granules of $G1$ that contain one or more granules of $G2$
- ▶ $\text{FirstWeekOfMonth} = \text{Select-up}(\text{FirstDayOfMonth}, \text{Week})$



Granularity Operations/7

- ▶ Combine(G_1, G_2)
- ▶ Combine all the granules of G_1 into one granule if they are contained within one granule of G_2 .
- ▶ $\text{BusinessMonth} = \text{Combine}(\text{BusinessDay}, \text{Month})$



Granularity Operations/8

- ▶ $\text{union}(G1, G2)$
- ▶ union, difference, intersection: the new granularity is the union, difference, intersection of the input granules
- ▶ Condition: if two granules of the two operands are non-disjoint (considering the underlying time) then they must be the same
- ▶ $\text{WeekendDay} = \text{union}(\text{Sunday}, \text{Saturday})$
- ▶ $\text{MondayAndFirstDayOfMonth} = \text{intersect}(\text{Monday}, \text{FirstDayOfMonth})$

Calendars/1

- ▶ A calendar is a collection of granularities that
 - ▶ is generated from a single bottom granularity, and
 - ▶ defines all non-bottom granularities in terms of granularity operations
- ▶ **Calendars** define granularities and determine the mapping between human-meaningful/readable time values and an underlying time line
 - ▶ e.g., the Gregorian Calendar defines the granularities second, minute, hour, day, week, fortnight, month, year, and decade.
 - ▶ e.g., “December 9, 1921” in the Gregorian calendar represents a specific set of time line chronons (a segment of the time line).

Calendars/2

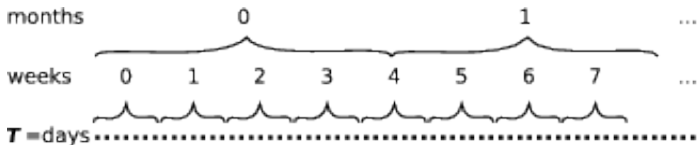
- ▶ Calendars incorporate the cultural, legal, and even business orientation of the user to define the time values that are of interest, e.g.,
 - ▶ Gregorian calendar
 - ▶ Business calendar
 - ▶ Useful calendar for tax or payroll applications
 - ▶ Days are the same as in the Gregorian calendar, but the Business calendar has a five day (work) week
 - ▶ The Business calendar year is divided into four quarters (Fall, Winter, Spring, Summer)
 - ▶ For tax purposes, the Business calendar year starts with the Fall quarter
 - ▶ Astronomy calendar
 - ▶ A year has 365.25 days
 - ▶ A century is precisely 36525 days long
 - ▶ Origin is noon on January 1, 4713 B.C.
 - ▶ The Gregorian calendar date “June 24, 1994” is 2449527.5 in the Astronomy calendar

Expressiveness and Computability

- ▶ Expressiveness of calendar algebra: able to express *finite*, *periodic*, or *semi-periodic* sets (a set is semi-periodic if there is a finite non-periodic start and an infinite periodic end)
- ▶ Computability is an important aspect. Has not been considered in depth.
- ▶ Syntactic restrictions have been identified that ensure computability
 - ▶ group into
 - ▶ finer
 - ▶ partition
 - ▶ sub-granularity
 - ▶ label-aligned

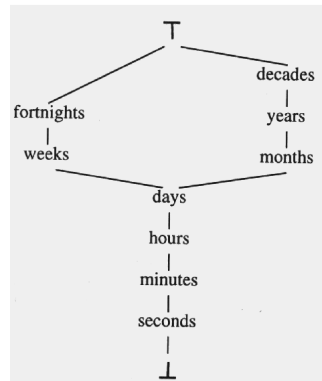
Lattice of Granularities/1

- ▶ Within a calendar, granularities are related in the sense that one granularity may be a **finer partitioning** of another
 - ▶ e.g., days are a finer partitioning of months or weeks
 - ▶ weeks are not a finer partitioning of months



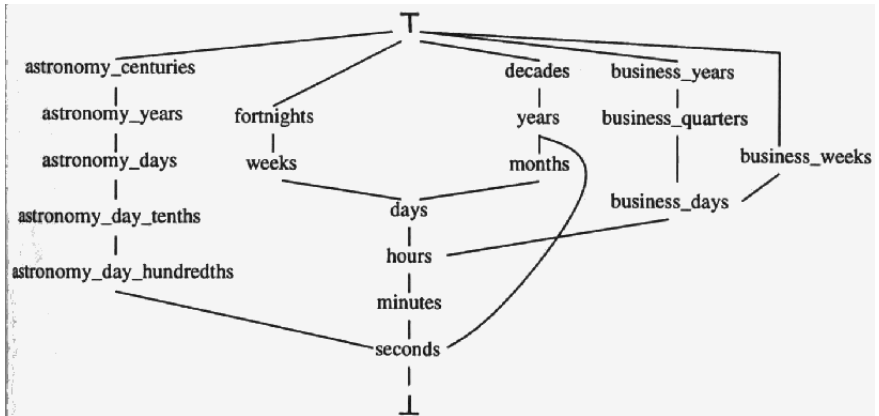
Lattice of Granularities/2

- ▶ With respect to **finer** partitioning, a set of granularities forms a lattice.
 - ▶ The top element, $\top$, is the maximal granularity of, i.e., the entire time-line
 - ▶ The bottom element, $\perp$, is the granularity of time-line clock (chronons)



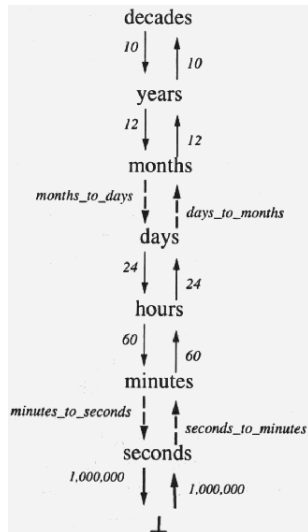
Lattice of Granularities/3

- ▶ A multi-calendar system
 - ▶ Granularities in different calendars are woven together into a single lattice



Lattice of Granularities/4

- ▶ Mappings between different granularities in a lattice have to be provided plus an anchor point
 - ▶ Regular versus irregular mappings
 - ▶ Complete versus incomplete mappings/partitioning
 - ▶ The properties of the mapping decide about efficient algorithms.



Granule Conversion/1

- ▶ A *granule conversion* converts granules in one granularity to granules in another granularity.
- ▶ We need to convert granules in order to process data measured at different granularities
- ▶ The *up conversion* maps the labels of a finer granularity G into the labels of a coarser granularity H : $\text{convert-up}(i, G, H) = \{j \mid G(i) \subseteq H(j)\}$.
- ▶ $\text{convert-up}(1, \text{day}, \text{month}) = 1$
- ▶ $\text{convert-up}(2, \text{day}, \text{month}) = 1$
- ▶ $\text{convert-up}(32, \text{day}, \text{month}) = 2$
- ▶ The *down conversion* maps the labels of a coarser granularity G into the labels of a finer granularity H : $\text{convert-down}(i, G, H) = \{j \mid G(i) \supseteq H(j)\}$.
- ▶ $\text{convert-down}(1, \text{month}, \text{day}) = \{1, 2, \dots, 31\}$
- ▶ $\text{convert-down}(2, \text{month}, \text{day}) = \{32, 33, \dots, 59\}$

Granule Conversion/2

- ▶ Granularity conversions are used to
 - ▶ Find the week of a particular day
 - ▶ Find the first Monday of a particular month
 - ▶ Find the last day of a particular fiscal year
 - ▶ Find the moon phase of a particular day
- ▶ There is always a common ancestor granularity that the source and target granularities can be defined on (the bottom granularity qualifies but more efficient ones might exist).
- ▶ A granularity conversion consists of two steps:
 - ▶ Convert the source granule to a set of granules in the ancestor granularity (down conversion).
 - ▶ Convert the granules from step 1 to granules of the target granularity (up conversion).

Cast Function

Current database systems provide the CAST function for conversions:

- ▶ **Cast** function $\text{cast}(T, G)$
 - ▶ Convert a timestamp T into granularity level G
 - ▶ Uses the mappings between different granularities

- ▶ Examples

```
cast( '1994-06-01', century) = '20'
cast( '1994-06-01', year) = '1994'
cast( '1994-06-01', month) = '1994-06'
cast( '1994-06-01', day) = '1994-06-01'
cast( '1994-06-01', hour) = '1994-06-01 00'
cast( '1994-06-01', minute) = '1994-06-01 00:00'
```

- ▶ Conversion from coarser to finer granularity
 - ▶ The cast function always chooses the first granule from the set of granules corresponding to the coarser timestamp
 - ▶ This avoids indeterminate results
- ▶ **Scale** function is similar, but produces an indeterminate result (a set of granules) when converting from a coarser to a finer granularity.

Instants, Periods, and Intervals/1

► Modeling Instants

- An **instant** is a point on the time line which is modeled by an **instant timestamp** that stores the number of a granule
- An instant timestamp records that an instant is located sometime **during** that particular granule
- The exact instant represented by an instant timestamp is never precisely known; only the granule during which it is located is known.
 - Two instants represented by the same granule might be different
- An instant is a point on a time-line, whereas a granule is a (short) segment of a time-line
- We assume that chronons, which are the smallest possible granule, are still bigger than instants.
- Distinction between chronons and instants captures the reality of measurements
 - All measurements are imprecise with respect to instants
 - We simply cannot measure individual instants: instants are “too small”.

Instants, Periods, and Intervals/2

► Modeling Periods

- A **period** is a duration of time that is **anchored** between two instants and is modeled by a **period timestamp**
- A period timestamp is the composition of two instant timestamps, where the start timestamp precedes or is equal to the end timestamp.
- We assume that the starting and ending timestamp are at the same granularity level

Instants, Periods, and Intervals/3

► Modeling Intervals

- An **interval** is an **unanchored** duration of time and is modeled by an **interval timestamp**
- The length of an interval is known, but not its starting or ending instants.
- An interval timestamp is a count of granules, e.g., 6 days.

Now

- ▶ An English noun meaning “*at the present time*”
 - ▶ A **distinguished timestamp value** in many temporal data models
 - ▶ Is a time instant rather than an interval or period
 - ▶ Reserved words for *now*: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, UC
 - ▶ Treated as a variable that is assigned a specific time during query or update evaluation.
 - ▶ As time advances, the interpretation of *now* also changes to reflect the new current time.
 - ▶ Common use of *now*
 - ▶ Indicate that a fact is valid until the current time (or until changed)
 - ▶ Jane began working as a faculty member for State University on 06/01/94
- | Name | Rank | ValidTime |
|------|-----------|------------------|
| Jane | Assistant | [06/01/94 – now] |
- ▶ Jane is a faculty member until we learn otherwise
 - ▶ Why use *now*
 - ▶ If the ground time were used, the terminating time of tuples that continue to be valid has to be updated as time advances
 - ▶ How to identify these tuples would be a costly process!

Summary

- ▶ Understand properties, strengths and limitations of standard relational database systems.
- ▶ Understand the limitations of relational database systems for modeling time-varying information:
 - ▶ coalescing
 - ▶ join
- ▶ Modeling of real world granularities
- ▶ Definition of calendars