Temporal Databases

Fabio Grandi
fabio.grandi@unibo.it
DISI, Università di Bologna

A short course on Temporal Databaes for DISI PhD students, 2016
Credits: most of the materials used is taken from slides prepared by Prof. M. Böhlen (Univ. of Zurich, Switzerland)

Applications of TDBs

- There are many examples of applications where some aspect of time is needed to maintain the required information in a DB:
  - ***Health care*: patient and treatment histories need to be maintained**
  - ***Insurance*: claims and accident histories are required**
  - ***Finance*: stock price and exchange histories need to be maintained**
  - ***Personnel management*: salary and position histories need to be maintained**
  - ***Banking*: account transactions and credit histories need to be maintained**

TDBs: What, When & Why

- Temporal databases, encompass all DB applications that require some aspect of time when organizing their information
- TDB applications have been developed since the early days of database usage. However, in creating these applications, it was mainly left to the developers to discover, design, program, and implement the temporal concepts
- They exhibit the need for developing *a set of unifying concepts and tools* for application developers to use

Limitations of Traditional DBs

- Modifications and deletions of data in a traditional DB are destructive (previous states are lost)

- Emp

Name	Dept	Salary
Tom	SE	2300
Ann	DB	3200

Emp

Name	Dept	Salary
Ann	DB	3400

- UPDATE Emp
  SET Salary = 3400
  WHERE Name = 'Ann';
- DELETE Emp
  WHERE Name = 'Tom'

No trace of the previous salary of Ann is maintained

No memory of an employee named Tom is kept

Limitations of Traditional DBs

- Traditional databases are snapshot, i.e. only the most recent state of the modeled reality is represented
- In order to maintain data histories, time columns can be added to the relation schema (tuple timestamps)
- Emp

Name	Dept	Salary	Start	End
Tom	SE	2300	1/1/12	1/1/16
Ann	DB	3200	1/1/10	1/1/15
Ann	DB	3400	1/1/15	Now

Limitations of Traditional DBs

- However, implementing a temporal data model is much more that adding a couple of columns to a table

  (e.g. there is no support for temporal integrity constraints, including the notion of a temporal key)

- Morever, the SQL query language provides very limited support for expressing temporal queries

  (e.g. temporal queries like a temporal join are very complex and error-prone to express in plain SQL)

A case study: Temporal join

Given the two relations

Emp

Name	Dept	Salary	Start	End
Tom	SE	2300	1/1/12	1/1/16
Ann	DB	3200	1/1/10	1/1/15
Ann	DB	3400	1/1/15	Now

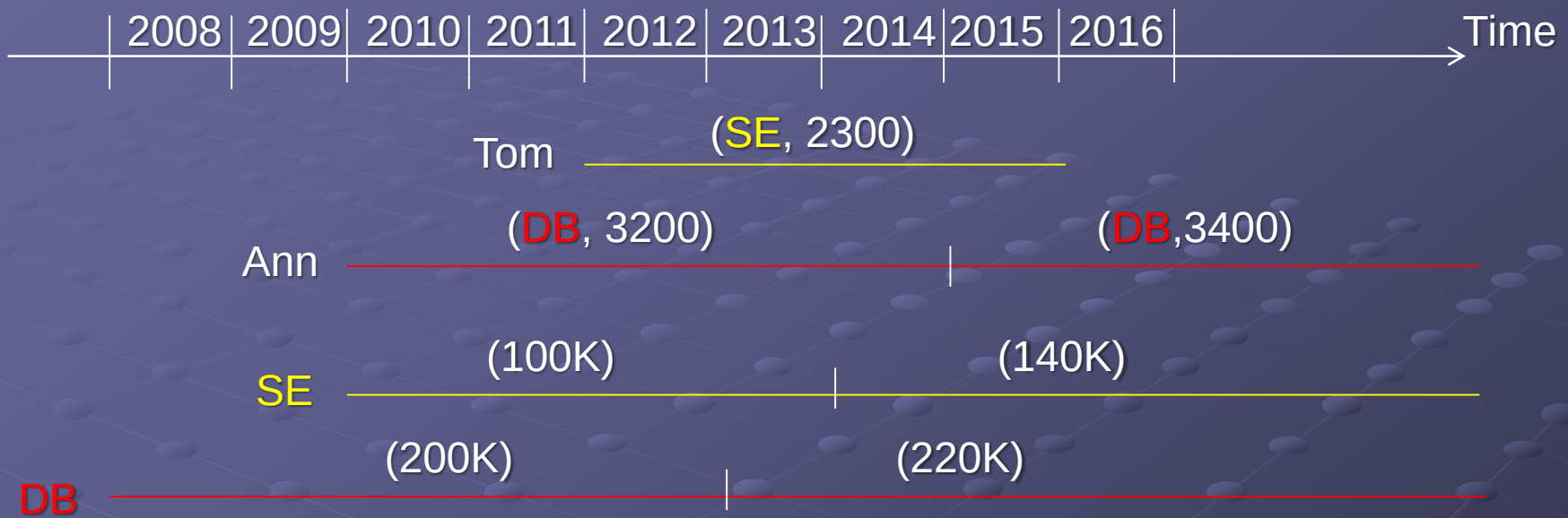
Dept

DName	Budget	Start	End
SE	100K	1/1/08	1/1/14
SE	140K	1/1/14	Now
DB	200K	1/1/10	1/1/13
DB	220K	1/1/13	Now

For each employee, find the history of the budget of the department they worked in

(two tuples join iff they match on the non temporal attributes and their timestamps overlap)

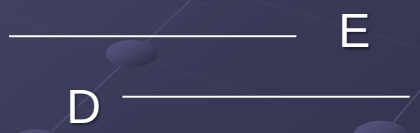
A case study: Temporal join



Name	Dept	Budget	Start	End
Tom	SE	100K	1/1/12	1/1/14
Tom	SE	140K	1/1/14	1/1/16
Ann	DB	200K	1/1/10	1/1/13
Ann	DB	220K	1/1/13	1/1/15
Ann	DB	220K	1/1/15	Now

A case study: Temporal join

```
SELECT Name, Dept, Budget, E.Start, E.End
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND D.Start <= E.Start AND E.End <= D.End
UNION
SELECT Name, Dept, Budget, E.Start, D.End
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND E.Start > D.Start
AND D.End < E.End AND E.Start < D.End
UNION
SELECT Name, Dept, Budget, D.Start, E.End
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND D.Start > E.Start
AND E.End < D.End AND D.Start < E.End
UNION
SELECT Name, Dept, Budget, D.Start, D.End
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND D.Start >= E.Start AND D.End <= E.End
```



A case study: Temporal join

Name	Dept	Budget	Start	End
Tom	SE	100K	1/1/12	1/1/14
Tom	SE	140K	1/1/14	1/1/16
Ann	DB	200K	1/1/10	1/1/13
Ann	DB	220K	1/1/13	1/1/15
Ann	DB	220K	1/1/15	Now

If we give up to time values in the result:

```
SELECT Name, Dept, Budget
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND D.Start <= E.End AND E.Start <= D.End
```

A case study: Temporal join

With CASE statements to compute timestamp intersection (if overlap [S1,E1] ∩ [S2,E2] = [max{S1,S2},min{E1,E2}]):

```
SELECT Name, Dept, Budget,
        CASE WHEN D.Start < E.Start
                THEN E.Start ELSE D.Start
        END,
        CASE WHEN D.End < E.End
                THEN D.End ELSE E.End
        END
FROM Emp AS E, Dept AS D WHERE Dept = DName
AND D.Start <= E.End AND E.Start <= D.End
```

A case study: Temporal join

Name	DName	Budget	Start	End
Tom	SE	100K	1/1/12	1/1/14
Tom	SE	140K	1/1/14	1/1/16
Ann	DB	200K	1/1/10	1/1/13
Ann	DB	220K	1/1/13	1/1/15
Ann	DB	220K	1/1/15	Now

The last two tuples are value-equivalent and could be coalesced (but how can it be done with SQL?):

Name	DName	Budget	Start	End
Tom	SE	100K	1/1/12	1/1/14
Tom	SE	140K	1/1/14	1/1/16
Ann	DB	200K	1/1/10	1/1/13
Ann	DB	220K	1/1/13	Now

Coalescing in pure SQL92

Coalesce the relation with schema R(X,S,E):
X=non temporal part; S=Start; E=End;

Use nested NOT EXISTS for universal quantification

Search for two (possibly the same)
value-equivalent tuples F (first) and L (last)

Ensure that there are no “time holes” between F.S and L.E
i.e. all start points M.S of value-equivalent
tuples M (mid) are extended (towards F.S)
by another value-equivalent tuple A1

Ensure that the interval between F.S and L.E is maximal
i.e. check via hypothetical value-equivalent tuple A2

Coalescing in pure SQL89

```
SELECT DISTINCT F.X, F.S, L.E
FROM R F, R L
WHERE F.S < L.E AND F.X = L.X
AND NOT EXISTS
(  SELECT *  FROM R M
   WHERE M.X = F.X  AND F.S < M.S AND M.S < L.E
    AND NOT EXISTS
    ( SELECT *  FROM R A1
       WHERE A1.X = F.X  AND A1.S < M.S AND M.S <= A1.E ) )
AND NOT EXISTS
(  SELECT *  FROM R A2
    WHERE A2.X = F.X
     AND ( A2.S < F.S AND F.S <= A2.E
              OR  A2.S <= L.E AND L.E < A2.E )  )
```

Limitations of Traditional DBMSs

- Traditional (non-temporal) DBMSs provide inadequate support for temporal aspects:
  - The data model and query language are basically “snapshot”
  - No built-in facilities for temporal integrity constraints (e.g. enforcement of a temporal key) are available
  - Temporal queries are very difficult to express and understand if expressed with non-temporal SQL
  - No support for the execution of temporal queries is provided in the query engines nor temporal access structures are available (traditional solutions reveal themselves inefficient)

In a temporal DBMS:

- The data model more accurately reflects the reality
- Temporal attributes might have a special semantics
- Queries shall be simpler with SQL temporal extensions

History of TDB research

We can distinguish 4 overlapping phases:

- 1956–1985: **Concept development,** considering the multiple kinds of time and conceptual modeling
- 1978–1994: **Design of query languages**
- 1988–present: **Implementation aspects,** including storage structures, operator algorithms, and temporal indexes
- 1993–present: **Consolidation phase**

  - Consensus glossary of temporal database concepts
  - TSQL2 language design initiative
  - Query language test suite

The Turning Point

- In the early 90's Temporal Databases were already a quite consolidated research field:
  - Around 800 scientific papers published
  - Around 100 scientists active in the field
  - Topic present in all main DB conferences
- The question then was: although temporal features are a recognized application requirement and lots of theoretical and practical technical contributions are available, *why temporal features are still completely absent in current commercial DB systems?* (and in the standard SQL92)
- One answer had to be found in the large diversity of the solutions proposed in the literature

The TDB Workshop

- A milestone in the settlement of the matter has been the ARPA/NSF TDB Workshop held in Arlington, TX in 1993
- The main artificer of the workshop and of many other things were happening has been Richard T. Snodgrass of the University of Arizona at Tucson (already well known in the community for his pioneering works in the field)
- The idea was to put together for a big 3-day brainstorming representatives of all the research groups involved in the field, in order to try to build consensus and draw the directions of future research efforts for the next decades

The Pre-workshop Initiatives

- The TDB workshop was preceded by some preliminary initiatives coordinated by Rick Snodgrass, involving remote cooperation between researchers spread all over the world and communicating via email
- The Temporal Database Glossary (e.g., to have a common language on which understand and frame different research solutions)
- The Temporal Query Test Suite (e.g., to have a shared benchmark on which compare the expressiveness and user-friendliness of different temporal languages)
- These remote cooperation efforts were also the feasibility testbed for the future TSQL2 design initiative

The Consensus Glossary Effort

- The effort was initiated in early 1992: a first embryo of the glossary was published on SIGMOD Record in September 1992 and as final chapter of the first book on Temporal Databases (Clifford, Tansel, Gadia, Jajodia, Segev & Snodgrass Eds.) published by Benjamin/Cummings in 1993

- The first mature version of the “Consensus Glossary” with 100 entries was included in the Workshop proceedings and discussed at the TDB Workshop (+ Addendum)

- All glossary entries were proposed, discussed, debated, refined and finally voted via email messaging; each individual who had contributed significantly were included as author of the glossary document

The Consensus Glossary Effort

- After the TDB Workshop, an editorial board was set up to supervise a revision of the glossary based on the input from the Workshop:

  James Clifford, Ramez Elmasri, Shashi K. Gadia, Pat Hayes, Sushil Jajodia and Christian S. Jensen

- A revised version was then made available as a TR and released to the general public via publication on SIGMOD Record in March 1994. The editorial board members appeared listed as editors before the other glossary authors (19)

The Consensus Glossary of TDB Concepts

- The effort was aimed at recommending standard definitions and names for concepts of common use within the TDB research community

- Two sets of criteria were defined:
  - All included concepts were judged against 4 relevance criteria (concepts specific, well-defined, well-understood, widely used)
  - Naming of concepts was resolved using 9 evaluation criteria (names orthogonal, easy to write, widely accepted, open-ended, without homonyms, conservative, consistent, intuitive, precise)

- Three categories of concepts were defined:
  - Of general database interest (e.g., valid or transaction time)
  - Of temporal database interest (e.g., temporal selection/projection)
  - Of specialized interest (e.g., temporally indeterminate)

The February 1998 Glossary Version

- In 1997, a Dagstuhl Seminar on TDBs was organized by Oren Etzion, Sushil Jajodia and Suri Sripada
- One of the by-products of the Seminar was the revision of the Consensus Glossary, also in the light of new available technical results and of the feedback received by the first published version from the academic and industrial worlds
- The revision, coordinated by Christian S. Jensen and Curtis E. Dyreson, produced a new version of the glossary that was included in the new TDB book coming out from the Seminar and published by Springer-Verlag in 1998

The Temporal Query Test Suite

- The goal was to provide a comprehensive consensus benchmark for temporal query languages
- The benchmark is semantic, and can be used to compare expressiveness and user-friendliness of different languages
- A full classification of queries was defined according to output/selection taxonomies; 3x10 query classes were defined and each contributing author was assigned a partition of classes to develop query examples
- The TSQL Benchmark has been published in the TDB Workshop proceedings and then included as a chapter in the TSQL2 book (with TSQL2 example queries)

The TSQL2 Initiative

- Rick Snodgrass circulated in May 1992 a white paper (TSQL: A Design Approach) soliciting the research community to join the efforts to develop a consensual proposal of temporal extension to the SQL92 standard query language
- After the TDB workshop, Rick proposed with an email to the workshop attendees the fulfillment of some standardization efforts involving the SQL92 temporal extension
- A TSQL2 Language Design Committee (the core thereof was made of the members of the TDB Workshop WG B) was set up, after a general invitation sent to the community

The TSQL2 Language Design Committee

- The TSQL2 Committee was made of 18 members including the coordinator, featuring representatives from universities, research and industrial labs:

  Richard T. Snodgrass (chair), Ilsoo Ahn, Gadi Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T.Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, Suryanarayana M. Sripada

- Starting from August 1993, all the language features were proposed, discussed, debated, refined and finally voted by all committee members via email messaging

The TSQL2 Language Specification

- A preliminary TSQL2 specification was released in January 1994 and a synopsis published in the March 1994 issue of ACM SIGMOD Record
- Also taking into account feedbacks received, a final specification (a 71-page TR) was released in in September 1994; a tutorial of the language was also published on SIGMOD Record
- The TSQL2 full specification, enriched with a collection of commentaries explaining the different aspects of the language design and other supporting materials, was publised as a book by Kluwer in 1995 (674+xxiv pages)

The TSQL2 Follow-Ups

- The direct successor of TSQL2, ATSQL [Böhlen, Jensen & Snodgrass 2000] introduced statement modifiers to override defaults and distinguish between sequenced and non-sequenced semantics of execution

- ATSQL became SQL/Temporal, a formal proposal of SQL92 temporal extension submitted in 1995 to the standardization committees to be included as Part 7 of the SQL3 standard under development

- Addition of valid-time and transaction-time were approved by ANSI in 1996 and forwarded to ISO at the beginning of 1997

The TSQL2 Follow-Ups

- The main criticism against TSQL2 and its successors involved the adoption of implicit timestamp columns and statement modifiers [Darwen & Date 2006]
- Hence, a “European” counterproposal, based on the temporal language IXSQL [Lorentzos & Mitsoupoulos 1997] supporting a generic interval data type, was also submitted to ISO in 1995
- Disagreements within ISO lead to cancellation of the temporal extension project but concepts and constructs from SQL/Temporal were then restored for SQL:2011

Recent History

- TDBs are 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 or GPS tracking to monitor employees, company cars and equipment)
  - data streams
  - data warehousing
- TDB techniques extended to other collateral fields (e.g. XML, Semantic Web)

Recent History

During recent years lots of efforts from companies:

- Oracle 9i, 2001: temporal extensions through workspace manager, time travel
- SAP HANA, 2010: history tables
- IBM DB2 10, 2010: Current and history tables, business time, system time, time travel
- Teradata 13.10, 2010: time travel, parts of ANSI SQL/Temporal
- SQL:2011 standard with temporal extensions

Third-party free or open-source tools are also available to add TDB facilities to mainstream DBMSs (incl. PostgreSQL, MySQL, SQL Server and Sybase)

Time Domains and Calendars

Time Domain

- Time domain/ontology
  - Specifies the building blocks of time
  - Time is generally modeled as an arbitrary set of instants/points with an imposed total order, e.g. (IN, ≤)
  - 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)
    - Partial order
    - 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

Time Domain

- Structure of time
  - Discrete time
    - Chronons (or temporal atoms, time quanta) are non-decomposable units of time with a positive duration
    - Chronon is the smallest duration of time that can be represented
    - Isomorphic to natural numbers
  - Dense Time
    - Between any two chronons another chronon exists
    - Isomorphic to rational numbers
  - Continuous time
    - Dense and no “gaps” between consecutive chronons
    - Chronons are durationless
    - Isomorphic to the real numbers

Time Domain

- Boundness of time
  - Time can be bounded in the past and/or in the future, i.e. first and/or last time instant exists
  - Time can be bound on one end (typically the past) and unbounded on the other end (typically the future)
- Relative (unanchored) versus absolute (anchored) time
  - “9 AM, January 1, 2016” is an absolute time
  - “9 hours” is a relative time (duration)

“Now”

- “Now” is a noun/adverb 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* (Until Changed)
  - Treated as a constant (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
- In the state-of-the-art
  - No DBMS allows to store NOW as a timestamp
  - There exist no solutions that do date computations with NOW

“Now”

- Common use of *now*
  - Indicate that a fact is valid until the current time (or until changed)
    - Ann began working in the DB department on 1/1/15

Name	Dept	Start	End
Ann	DB	1/1/15	Now

    - Ann is in the DB department 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 such tuples could be a costly process
  - Determining the duration of periods yields meaningful results (e.g. impossible if we would use a Null value instead)

Time Domain - Summarizing

- Humans perceive time as continuous
  and time is assumed continuous in classical physics

- A **discrete linear unbounded 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. 1:30 pm means over some period/chronon around this time
  - Chronons allow easily modeling of durative events
  - Any implementation needs a discrete encoding of time
  - Time keeps on growing without an upper bound

- It may be a problem to represent continuous evolution (e.g. movement) in a discrete model

Timestamps - Instants

- An instant is a point on the time line which is modeled by an instant timestamp that stores the number of a granule
  - e.g. SemesterStart(EngUniBO2, 22/2/2016)
  - WasBorn(Einstein, 1879)
- An instant timestamp records that an instant is located sometimes 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

Timestamps - Instants

- 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 mesaurements are imprecise with respect to instants
  - We simply cannot measure individual instants: instants are “too small”
  - We assign instants to the chronon that contains them
- Instant timestamps can be represented in a relational table with a single column

Timestamps - Periods

- A period is a duration of time that is anchored between two instants and is modeled by a period timestamp
  - e.g. Emp(John, Clerk, 1/6/2012 – 31/12/2013)
  - Seminar(TDB1, 5/2/16 10:00 – 5/2/16 13:00)

- A period timestamp is the composition of two instant timestamps, where the start precedes or is equal to the end
- We assume that the starting and ending timestamp are at the same granularity level
- We either use two instant timestamps S, E or a period timestamp (S-E, [S,E), [S,E]).
- Periods can be closed, half-open or open: [2003,2005], [2003,2005), (2003,2005)

Timestamps - Periods

- Instant timestamps can be represented in a relational table with two columns

- Closed to the left and open to the right intervals are usually assumed in TDBs

Name	Job	Start	End
John	Clerk	1/6/12	1/1/14

the timestamp of the fact “John worked as clerk”
is [1/6/2012, 1/1/2014) = [1/6/2012,31/12/2013],
i.e. the last day he worked as clerk is December 31, 2013

Timestamps - Elements

- A temporal element is a set of time periods
  - e.g. holiday(Jim, { 1/8/15 – 20/8/15, 20/12/15 – 8/1/16 })
- Mathematically, a temporal element is more attractive than a period because a closed algebra can be defined: subtraction and union of temporal elements yields a temporal element again (it does not with periods)
- In the real world temporal elements are used rarely

Timestamps - Intervals

- A temporal interval is an unanchored duration of time and is modeled by an interval timestamp
  - e.g. trip(Milan-SanFrancisco, 20 hours)
  - holidays(employee, 30 days)
- The length of an interval is known, but not its starting or ending instants
- An interval timestamp is a count of granules, e.g. 10 days

Periods versus Intervals

- In mathematics and physics, we define the period as the repetition interval of a periodic phenomena (e.g. sine and cosine functions have a period of 2π), i.e. as a pure duration
- What we defined as period is exactly what in mathematics is called interval (e.g. a time interval in physics)
- Our somehow “counterintuitive” definition of periods and intervals is due to their introduction as SQL92 datatypes
- TDB researchers are very sorry for this... ☹

Granularities

- The aim of the introduction of granularities is twofold:
  - Coarser granules are often more convenient than smaller granules, e.g. 20 years versus 7305 days
  - The exact date is not known at a smaller granularity, e.g. we know that the date is April 2015 but do not have an exact day
- The goal when defining granularities (and calendars) is to not enumerate all time points but to have a compact definition of real world granularities
- A compact definition can be used as a starting point for compact representations, efficient implementations, etc.
- We give an algebraic definition of natural granularities

Time Granularity

- 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 at granularity of hours (or half-hours)
  - train schedules at 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

- **Example:** Airline flight database

Departures

Flight	Time
100	2015-08-01 12:30
55	2015-09-10 11:15
256	2016-01-01 16:40

Vacations

Vacation	Time
Christmas	[2015-12-24, 2016-01-01]
Easter	[2015-04-02, 2015-04-07]
Summer	[2015-08-01, 2015-08,30]

- 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

Time Granularity

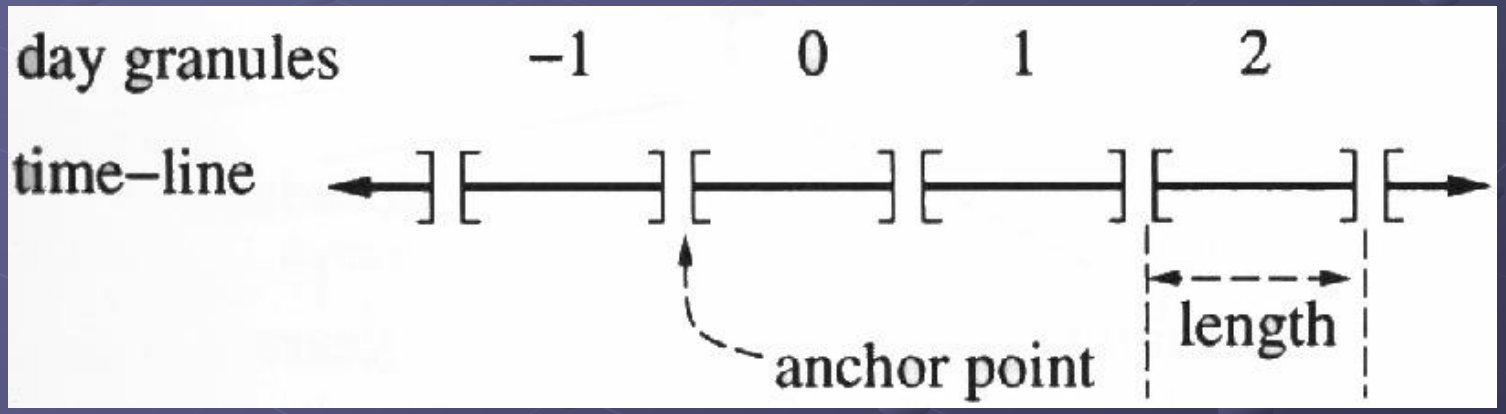
- Query: *Which flights left during the Christmas vacation?*

```
SELECT *
FROM Vacations V, Departures D
WHERE Vacation = ‘Christmas'
AND V.Time OVERLAPS D.Time
```

- *Problems:*
  - *Query processor needs to know the relationship between minutes and days*
  - *Is overlaps evaluated at granularity of days or at granularity of minutes?*

Time Granularity

- 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

Time Granularity

- 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. Week(2) = {8, 9, 10, 11, 12, 13, 14}

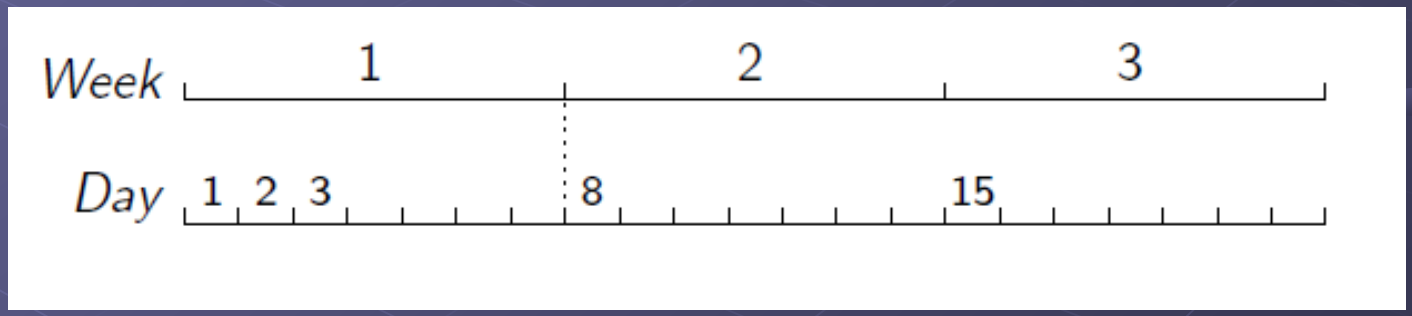
Time Granularity

- 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

- Group(G, StartIndex, NumGrans)
- Start at granule StartIndex and repeatedly group NumGrans granules into one granule

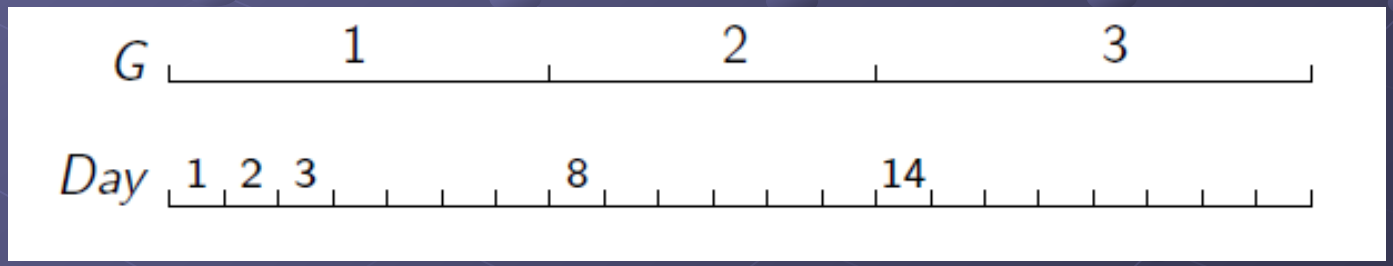
- Example:
  - Week = Group(Day,1,7)



Granularity Operations

- 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

- Example:
  - G = Alter(Day,Week,2,-1,3)

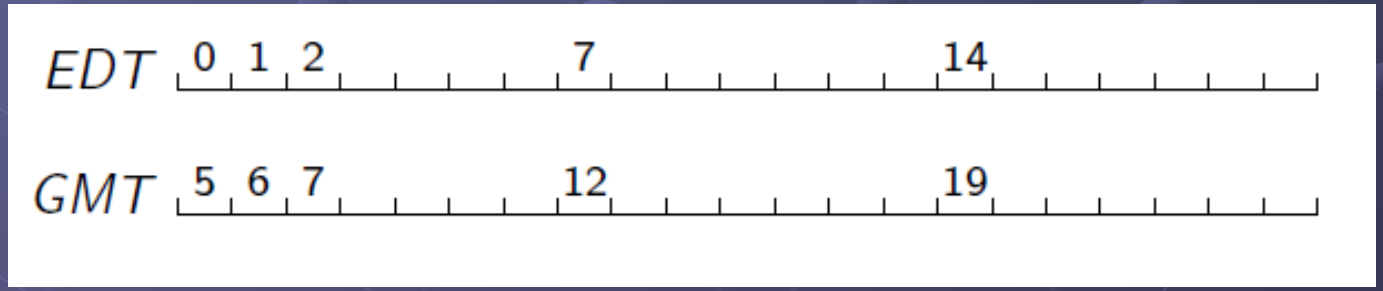


- Examples: leap years, leap seconds

Granularity Operations

- Shift(G,m)
- Shifting operation allows to shift the index set G by m positions

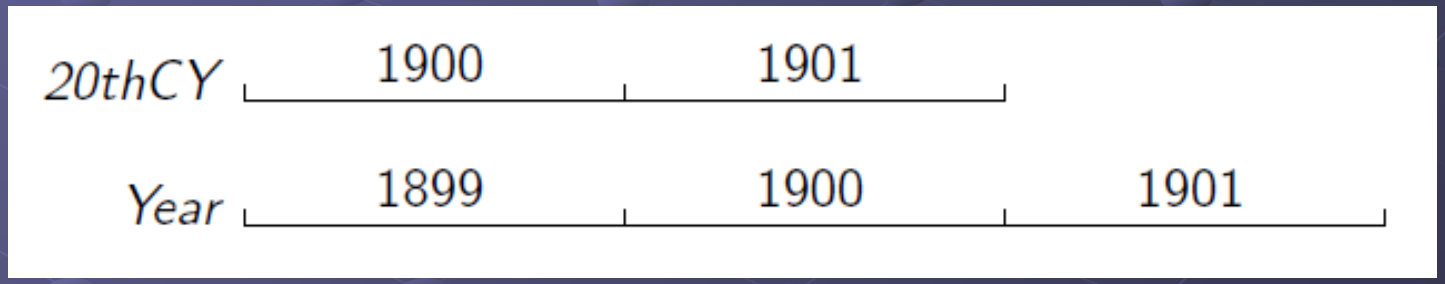
- Example:
  - EDT = Shift(GMT,5)



Granularity Operations

- Subset(G,m,n)
- Takes all the granules of G whose labels are in the interval from m to n

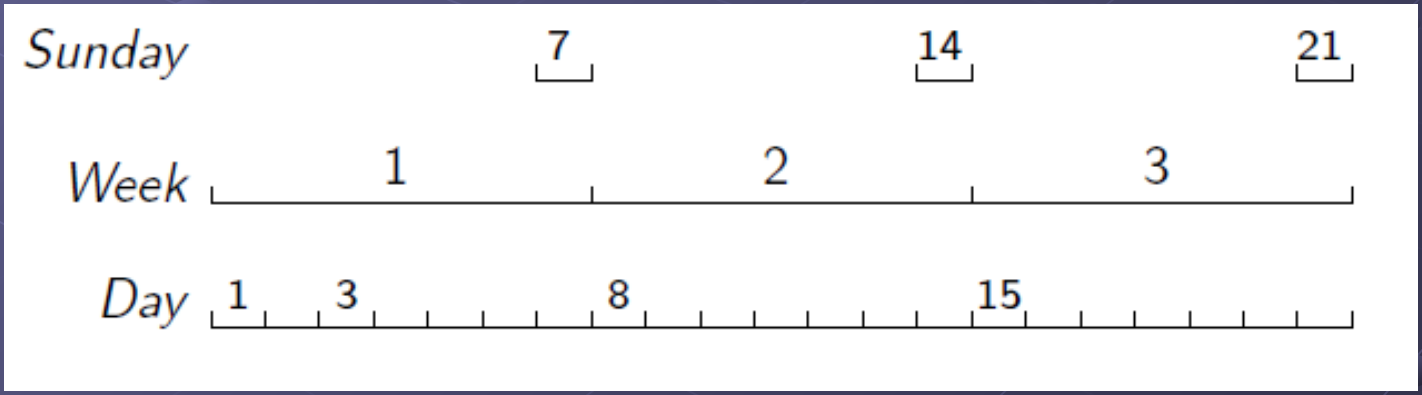
- Example:
  - 20thCenturyYears = Subset(Year,1900,1999)



Granularity Operations

- 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

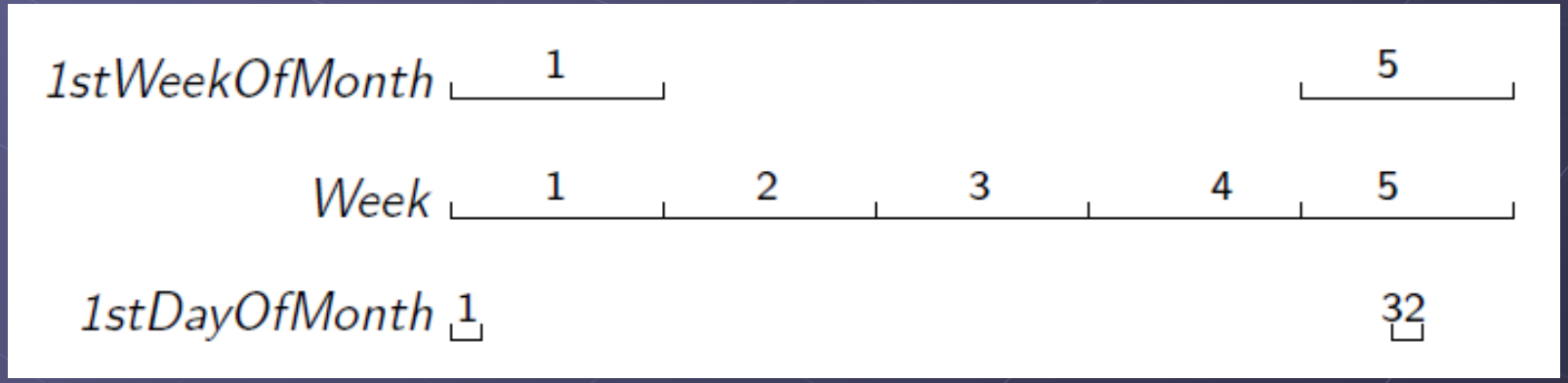
- Example:
  - Sunday = Select-down(Day,Week,7,1)



Granularity Operations

- Select-up(G1,G2)
- Selects the granules of G1 that contain one or more granules of G2

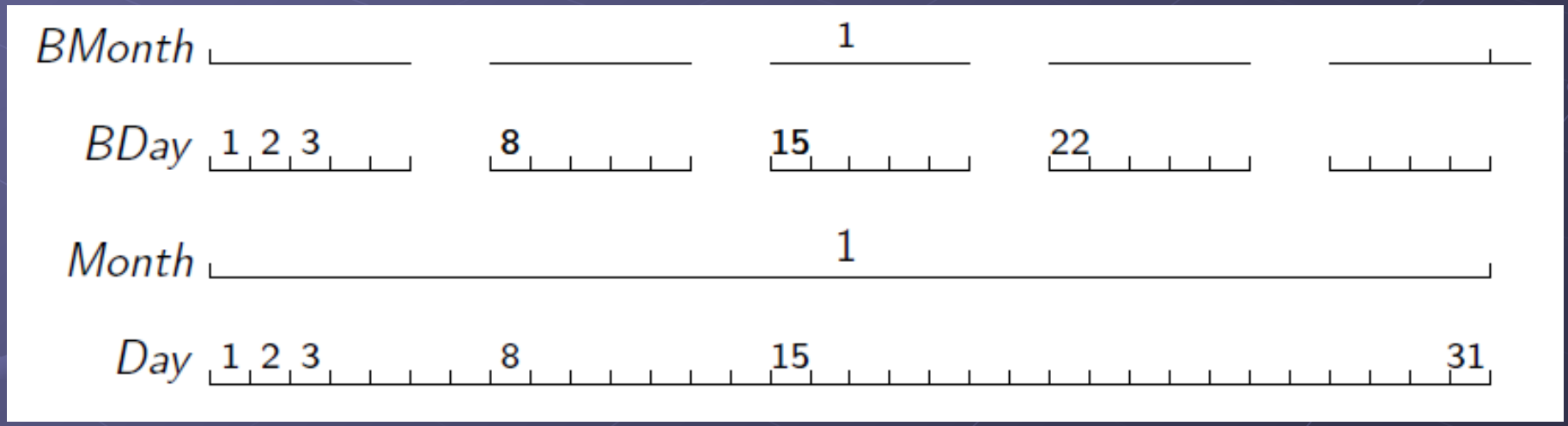
- Example:
  - FirstWeekOfMonth = Select-up(Week,FirstDayOfMonth)



Granularity Operations

- Combine(G1,G2)
- Combine all the granules of G1 into one granule if they are contained within one granule of G2

- Example:
  - BMonth = Combine(BDay,Month)



Granularity Operations

- Union(G1,G2), Difference(G1,G2), Intersection(G1,G2)
- 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

- Example:
  - WeekendDay = union(Sunday, Saturday)
  - MondayAndFirstDayOfMonth = intersect(Monday, FirstDayOfMonth)

Calendars

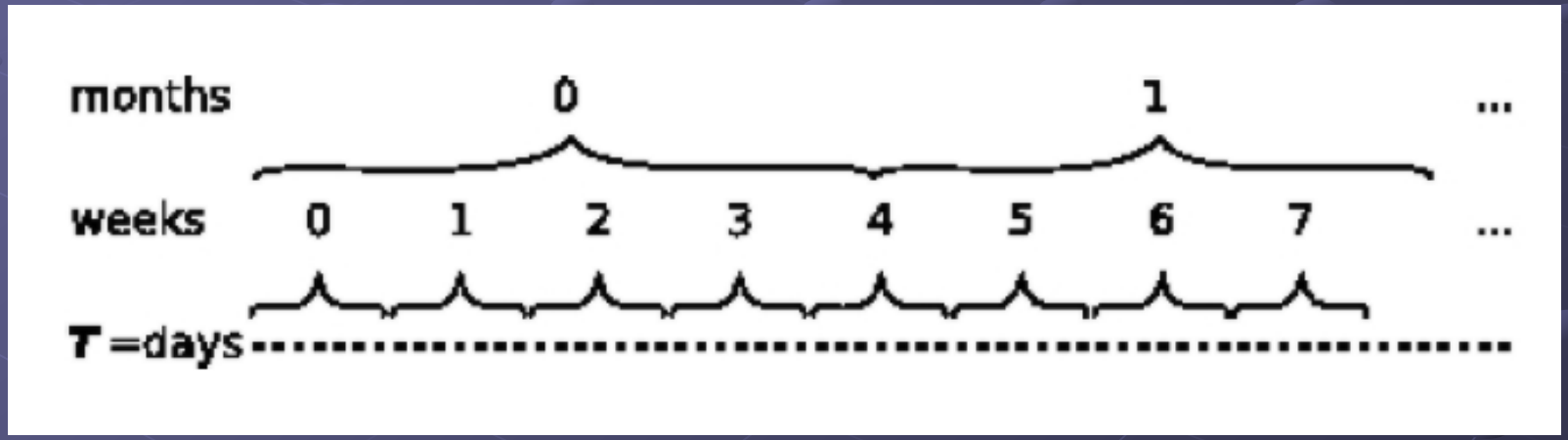
- A calendar is a collection of granularities that
  - is generated from a single bottom granularity, and
  - defines all non-bottom granularities in terms of granularity operations
- Calendars define granularities and determine the mapping between human-meaningful/readable time values and an underlying 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

- Calendars incorporate the cultural, legal, religious 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

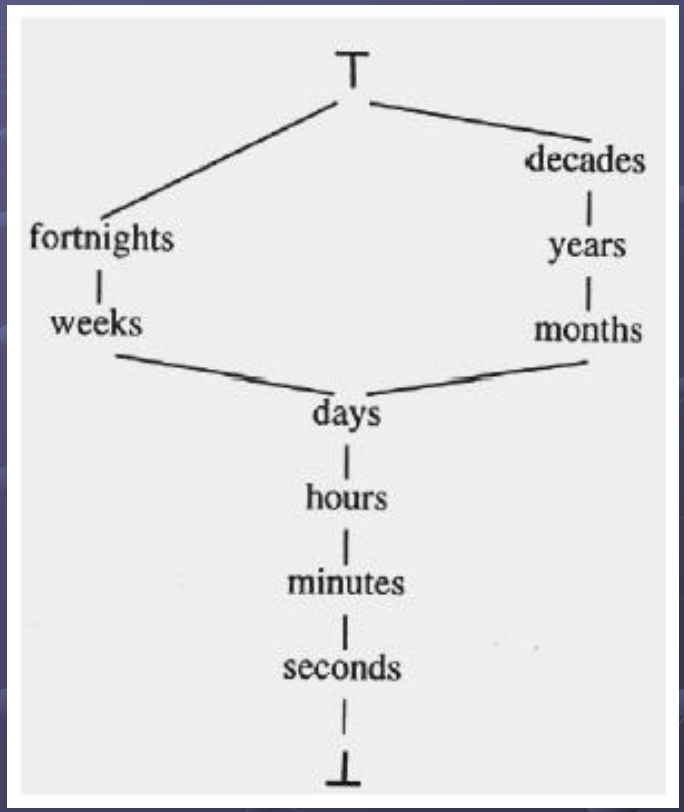
Lattice of Granularities

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



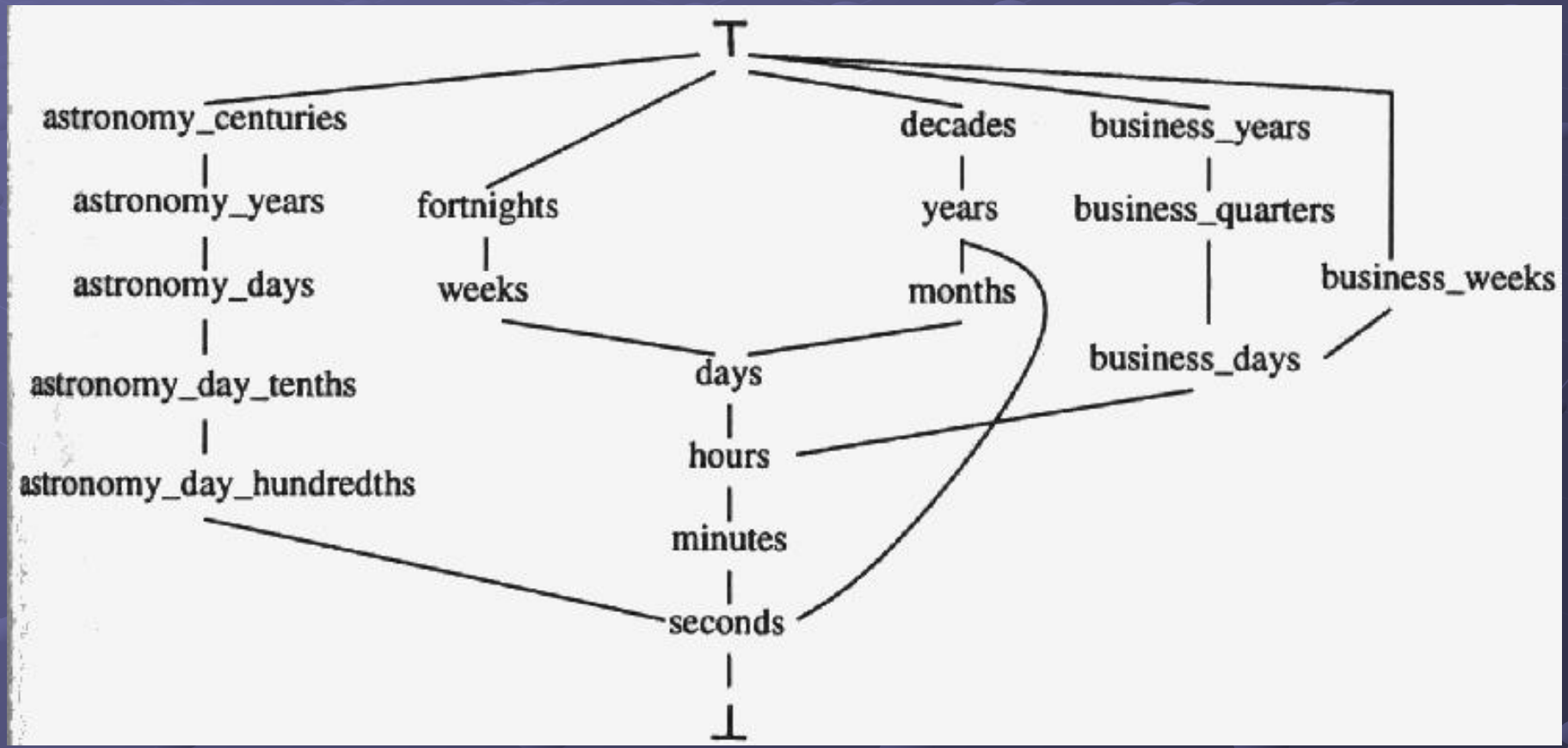
Lattice of Granularities

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



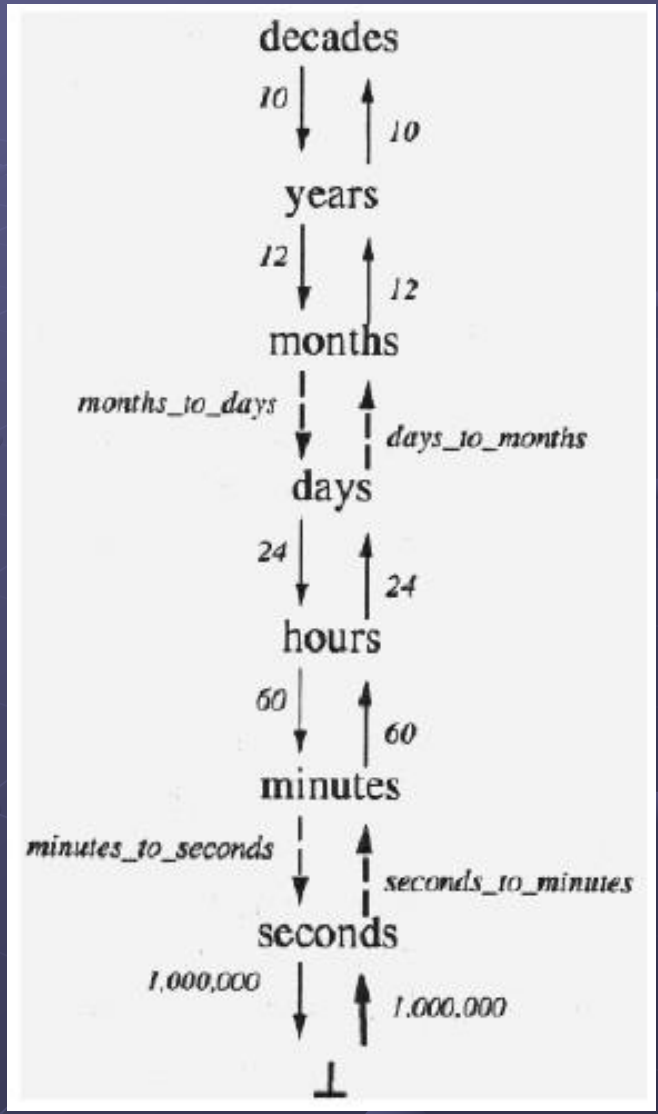
Lattice of Granularities

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



Lattice of Granularities

- 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

- A granule conversion converts granules in one granularity to granules in another granularity
- The up conversion maps the labels of a finer granularity G into the labels of a coarser granularity H:
  convert-up(i,G,H) = {j | G(i) ⊆ H(j)}
  - convert-up(1,day,month) = 1
  - convert-up(2,day,month) = 1
  - convert-up(32,day,month) = 2
- The down conversion maps the labels of a coarser granularity G into the labels of a finer granularity H:
  convert-down(i, G,H) = {j | G(i) ⊇ H(j)}
  - convert-down(1,month,day) = {1,2,...,31}
  - convert-down(2,month,day) = {32,33,...,59}

Granule Conversion

- We need to convert granules in order to process data measured at different granularities
- 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 *CAST*(*T, G*) to 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', DAY) = '1994-06-01'

  CAST('1994-06-01', HOUR) = '1994-06-01 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