## Temporal and Spatial Data Managament
## Fall 2017

# Time Domain and Calendars
## SL02

- Time domain and timestamps
- Time granularity
- Calendars



## Table of Contents





## 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)
    - 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/2

- **Density** of time
  - **Discrete time**
    - **Chronons** 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/3

- **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 unbound on the other end (typically the future)
  - The time domain includes a special constant for the current time.
- **Relative** (unanchored) versus **absolute** (anchored) time
  - "9 AM, January 1, 1996" is an absolute time
  - "9 hours" is a relative time



## Time Domain/4

- "Now" is an English 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`
  - 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.
- State of the art:
  - No DBMS allows to store NOW as a timestamp
  - There exist no solutions that do date computations with NOW



## Time Domain/5

- 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 the University of Arizona on 94/06/01

| Name | Rank | Time |
|---|---|---|
| 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 could be a costly process.
  - Determining the duration of periods yields meaningful results.



## Time Domain/6

Summarizing:

- Humans perceive time as continuous
- 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., 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
  - Time keeps on growing without an upper bound.
- Problem to represent **continuous movement** in a discrete model
  - An object is continuously moving from point A to point B

## Timestamps—Instants/1

- Examples:
  - emp(Joe, IfI, 2003)
  - SemStart(uzh, 2012/9/17)
- 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



## Timestamps—Instants/2

- 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".



## Timestamps—Periods

- Examples:
  - emp(Joe, IfI, 2003/7 - 2006/10)
  - SemStart(uzh, 2012/9/26 12:15, 2012/9/26 13:45)
- 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 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—Temporal Elements

- Examples:
  - holiday(Lena, $\{[2012/7/3, 2012/7/24], [2012/10/7, 2012/10/14]\}$ )
- A **temporal element** is a set of time periods.
- Mathematically, a temporal element is more attractive than a period because it is closed: subtraction and union of temporal elements yields a temporal element again.
- In the real world temporal elements are used rarely.

## Timestamps—Intervals

- Examples:
  - session(Tom, 3, 25 minutes)
  - trip(Zürich, Phoenix, 20 hours)
- 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.



## Table of Contents





## Granularities

- Granularities are introduced for two purposes:
  - Coarser granules are 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 May 2014 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/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 at granularity of 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/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



# Time Granularity/3

- Query: *Which flights leave during the Thanksgiving vacation?*

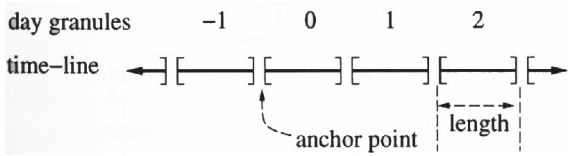
```
SELECT *
FROM Vacations V, FlightDepartures FD
WHERE Vacation = 'Thanksgiving'
AND V.Time OVERLAPS FD.Time
```

- Problems:
  - Query processor needs to know the relationship between minutes and days.
  - Why is *overlaps* evaluated at granularity of days rather than minutes?



# Time Granularity/4

- **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/5

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

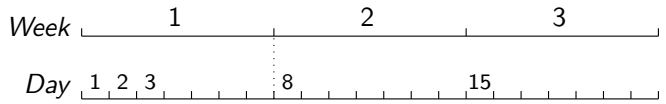
# Time Granularity/6

- **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

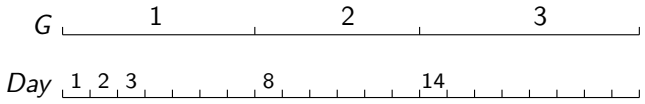
- Group(G, StartIndex, NumGrans)
- Start at granule StartIndex and repeatedly group NumGrans granules into one granule
- Example:
  - Week = Group(Day,1,7)






# Granularity Operations/2

- 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)

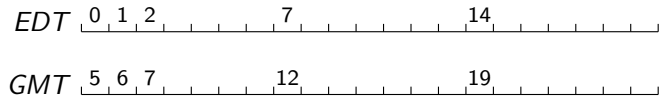



- Examples: leap years. leap seconds



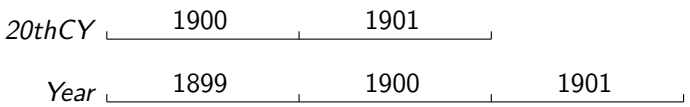
# Granularity Operations/3

- Shift(G,m)
- Shifting operation allows to shift the index set G by m positions
- Example:
  - EDT = Shift(GMT,5)



## Granularity Operations/4

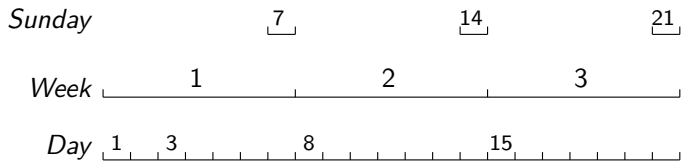
- Subset(G,m,n)
- Takes all the granules of G whose labels are in the interval from m to n
- Example:
  - 20thCenturyYears = Subset(Year,1900,1999)






## Granularity Operations/5

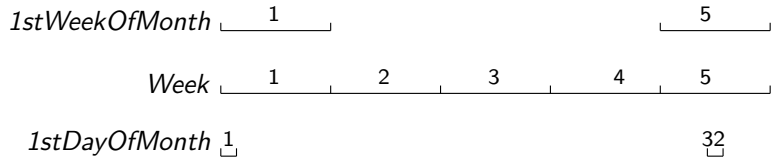
- Select-down(G1,G2,k,l)
- Selects granules of G1 by picking up l granules starting from the kth one in each set of granules of G1 contained in one granule of G2
- Example:
  - Sunday = Select-down(Day,Week,7,1)






## Granularity Operations/6

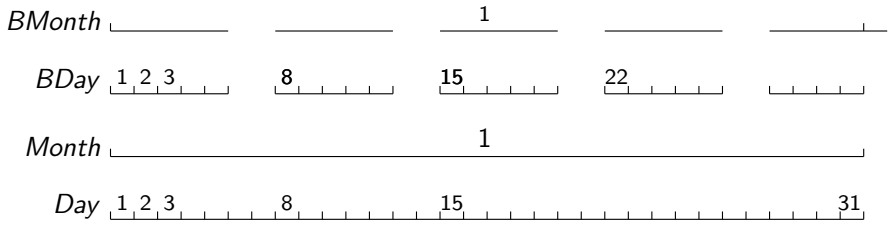
- Select-up(G1,G2,k,l)
- Selects the granules of G1 that contain one or more granules of G2
- Example:
  - FirstWeekOfMonth = Select-up(Week,FirstDayOfMonth)






## Granularity Operations/7

- 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/8

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



## Table of Contents





## Calendars/1

Task 2.2

- 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).

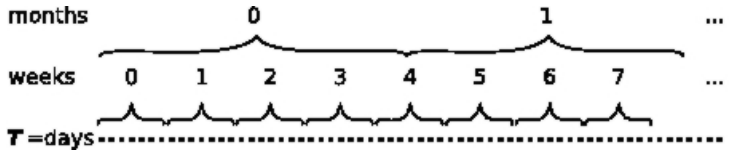


## Calendars/2

- 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

# Lattice of Granularities/1

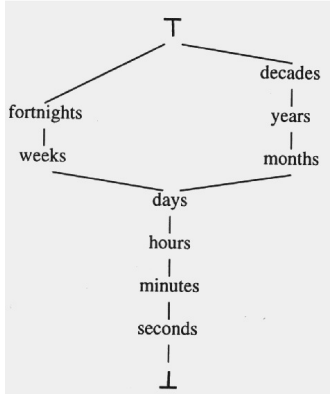
- Within a calendar, granularities are related in the sense that one granularity may be a **finer parititioning** of another
  - e.g., days are a finer partitioning of months or weeks
  - weeks are not a finer partitioning of months






# Lattice of Granularities/2

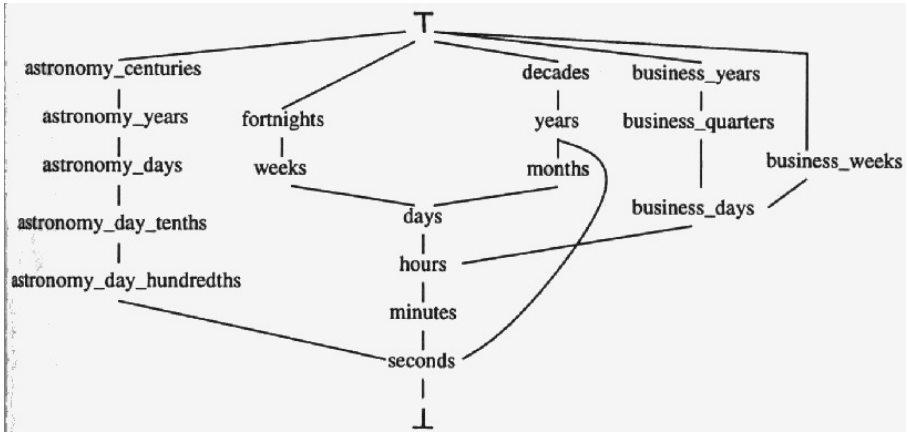
- With respect to **finer** partitioning, a set of granularities forms a lattice.
  - The top element, $\top$, is the maximal granularity of, i.e., the entire time-line
  - The bottom element, $\bot$, is the granularity of time-line clock (chronons)






# Lattice of Granularities/3

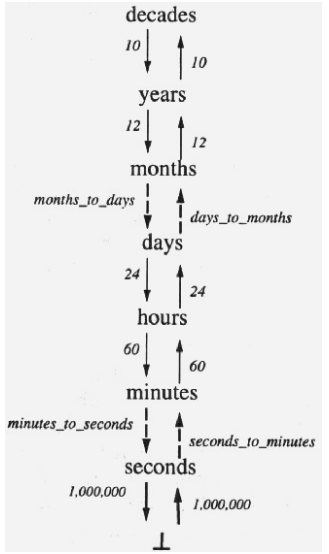
- A multi-calendar system
  - Granularities in different calendars are woven together into a single lattice






# 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.
- The *up conversion* maps the labels of a finer granularity $G$ into the labels of a coarser granularity $H$:
  convert-up$(i, G, H) = \{j \mid G(i) \subseteq H(j)\}$.
- 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$(i, G, H) = \{j \mid G(i) \supseteq H(j)\}$.
- convert-down(1,month,day) $= \{1,2,...,31\}$
- convert-down(2,month,day) $= \{32,33,...,59\}$



## Granule Conversion/2

- 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 for conversions:

- **Cast** function $cast(T, G)$
  - 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', 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.



## Summary

- Properties of the time domain
  - discrete, dense, continuous
  - bounded, unbounded
  - relative, anchored
  - Instants, periods, intervals
  - Now
- A discrete linear model is the default for temporal database systems
- Granularities are necessary abbreviations for sets of time points.
- A systematic approach (e.g., an algebra) is needed to manipulate granularities
- A calendar is a collection of granularities
- Granule conversions to change granularity