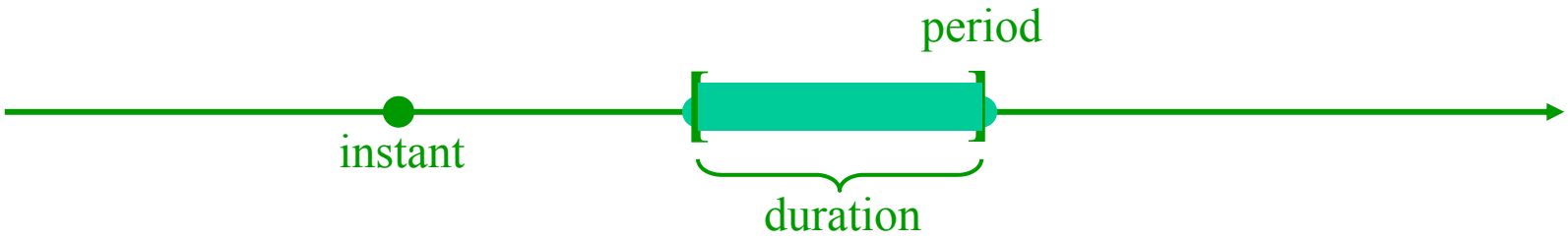
Representing Time in SQL

Chapter 2

Temporal Objects in Measured Time: A Quick Reminder

- From now on we will exclusively deal with measured time, i.e., with temporal objects on a discrete time axis using a calendar and/or a clock.

- There are three kinds of temporal objects (according to the „neutral“ terminology introduced in the previous chapter):
  - Instants – points on the time axis
  - Periods – intervals on the time axis
  - Durations – lengths of periods



- Examples (in natural language):
  - Instant: 22.4.2015 10:30 a.m.
  - Period: 22.4.2015 10:30 a.m. – 22.4.2015 12:00 a.m.
  - Duration: 90 minutes

Temporal Data Types in SQL: Overview

„SQL includes an extensive set of features for support of dates and times. Unfortunately, however, that support is quite complex (this remark is not intended as a criticism – the fact is, many aspects of dates and times are *inherently* complex, as is well known.)“

(from Date/Darwen „A Guide to the SQL Standard)

SQL offers a variety of temporal data types (with corresponding literals, operators, value functions, and predicates) that can be roughly sketched as follows:

DATE	
TIME	(WITH TIME ZONE)
TIMESTAMP	(WITH TIME ZONE)
INTERVAL	

The first three types (or better: type classes) refer to instants, whereas the last one refers to durations. The first three are also jointly called *datetimes* in many publications. There is still no temporal data type for periods in SQL!

Reminder from Chapter 1: Variants of Date Representation

"A **date** … is a reference to a particular day represented within a calendar system."

en.wikipedia.org: Calendar date

- There are three different conventions for ordering the three components in dates (according to the Gregorian calendar):
  - Day – Month – Year (DMY), e.g.: 22. April 2015
  - Year – Month – Day (YMD), e.g.: 2015, April 22
  - Month – Day – Year (MDY), e.g.: April 22, 2015

- Different regions in the world use different orderings, e.g.:
  - DMY is used in most of Europe, South America, southern Asia, Australia
  - YMD is used in northern Asia (China, Japan, Corea, Iran)
  - MDY is used in the USA

  (for details see *en.wikipedia.org: Date format by country*)

- For each component (D, M, Y) as well as for the delimiters between the components there are various representation variants, e.g. (in DMY style):

  22. April 2015, 22.4.2015, 22-Apr-2015, 22/04/2015

DATE Type in SQL

- Values of type DATE denote instants on the (measured) time line according to the Gregorian calendar with granularity DAY. Values are represented in SQL by literals.

- DATE literals are expressed in YMD style according to the convention of ISO 8601, consisting of
  - the qualifier DATE
  - followed by a 10-character string of the form 'yyyy-mm-dd'
    (with 4 digits for the year, and 2 digits each for month and day).

    e.g.: DATE '2012-04-23'

- Field values within a DATE literal are constrained as follows:

YEAR:	0001 to 9999
MONTH:	01 to 12
DAY:	01 to 31

- DATE values are further constrained by the rules of the Gregorian calendar, e.g., dates '1999-04-31' or '1990-02-29' are illegal.

- Thus, the range of dates is from '0001-01-01' to '9999-12-31'.

Reminder: Variants of Time Representation

- There are two different kinds of clocks in use in different regions of the world:
  - a 24-hours clock, where hours range from 00 to 23
  - a 12-hours clock, where hours range from 01 to 12, based on a division of the day into two periods: a.m. (latin: „ante meridiem“, before midday) and p.m. (lat.: „post meridiem“, after midday)

- The 12-hour clock is mainly used in English speaking countries, but is also in use as informal/colloquial alternative in many countries preferring the 24-hour system, e.g., in Germany.

- Again, there are different notation variants in use:
  - The hour component is either written with two digits (01, 02, ..., 12, ..., 24) or in mixed style with one or two digits (1, 2, ..., 12, ..., 24).
  - The delimiter between hour and minute is either a colon (:)or a full stop (.).
  - The period of day in 12-hour style is either written a.m./p.m. or AM/PM.

- Midnight is denoted by 00:00 in 24-hour style, not by 24:00! Midnight is denoted by 12:00 p.m. in 12-hour style, whereas 12:00 a.m. denotes midday – this is strictly speaking a contradiction (midday being after midday), but fits well with 12:01 a.m. In 12-hour notation, 12:59 a.m. is followed by 1:00 a.m.!

TIME Type in SQL

- In SQL, values of type TIME denote instants on the (measured) time line of a 24-hour clock with granularity SECOND.

- TIME literals follow ISO 8601 conventions, too, consisting of
  - the qualifier TIME
  - followed by an 8-character string of the form 'hh:mm:ss'
    (with 2 digits each, for the hour, the minute, and the second).

e.g.: TIME '12:45:02'

- Field values within a TIME literal are constrained as follows:

HOUR:	00 to 23
MINUTE:	00 to 59
SECOND:	00 to 61

- Time values in SQL are relative to Coordinated Universal Time (UTC), formerly called Greenwhich Mean Time (GMT). Occasionally, UTC has to be adjusted to astronomically determined time by insertion of 1 or 2 leap seconds. Thus, there may be minutes with 61 or 62 seconds. The last such leap second was inserted on 30.6.2015 at midnight.

TIME with Fractional Part

- Optionally, TIME values may be used with lower granularity (i.e., higher precision) than second (e.g., milliseconds, i.e., 10^{-3} seconds).

- In order to do so, a fractional part may be added to TIME literals , separated from an „ordinary“ time literal by a dot, e.g.:

```
TIME(3) '19:12:01.250'
```

(i.e., 19:12:01 plus $250 * 10^{-3}$ seconds)

- The number of digits in the fractional part is to be added to TIME in parentheses, i.e., there are variant TIME types such as TIME(1), TIME(2), ..., TIME(6).

- The default value – corresponding to TIME – is TIME(0).

- Just for reminding you:

10^{-3} s = 1 ms (millisecond)
10^{-6} s = 1 µs (microsecond)
10^{-9} s = 1 ns (nanosecond)

(There are various other special names for second fractions.)

TIMESTAMP Type in SQL

- Values of type TIMESTAMP denote instants on the (measured) time line according to the Gregorian calendar with granularity microsecond, combining DATE and TIME(6).

- TIMESTAMP literals consist of
  - the qualifier TIMESTAMP
  - followed by a 26-character string of the form 'yyyy-mm-dd hh:mm:ss.nnnnnn' (which is a combination of a DATE and a TIME(6) literal separated by a space).

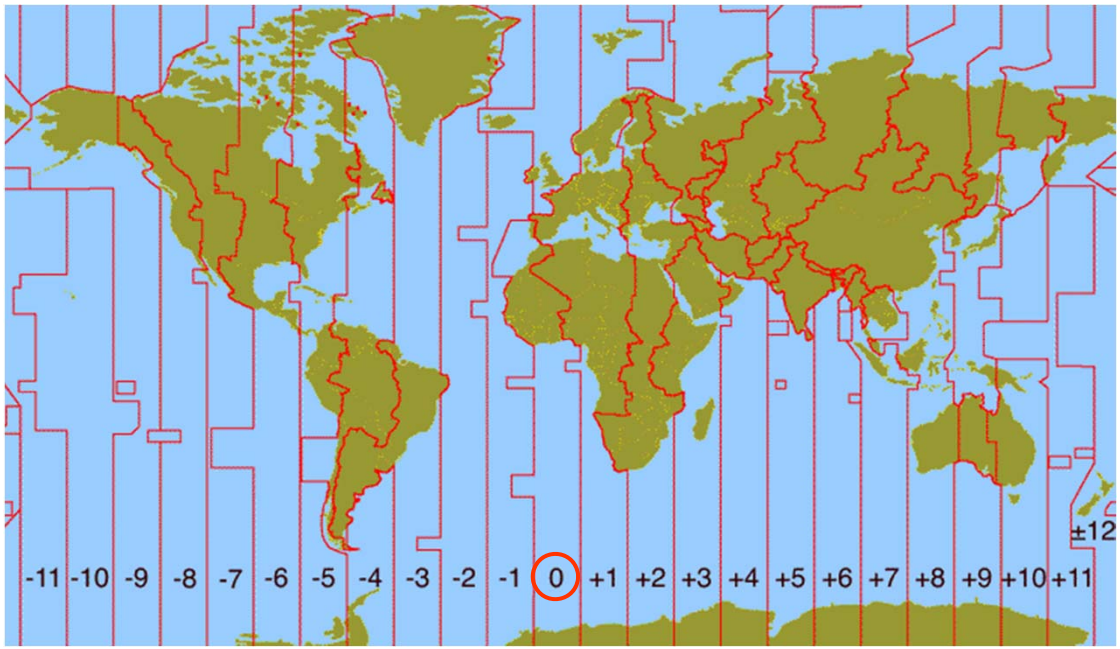
e.g.: TIMESTAMP '2011-05-02 12:45:02.000000'

- There are variant types with lower or higher granularity than microsecond, too, expressed by indicating the length of a fractional part (if any) as in TIME, e.g., TIMESTAMP(4).

- However, the default precision in case of TIMESTAMP is 6 (rather than 0 as for TIME)! So if you want to use „normal“ second granularity, as in TIME, for TIMESTAMPs, you have to specify type TIMESTAMP(0), rather than simply using TIMESTAMP.

Time Zones and UTC: Just a Reminder

UTC



WITH TIME ZONE

- For TIME and TIMESTAMP there are type variants with explicit indication of the time zone – if no such indication is given, UTC is implicitly assumed:

```
TIME      WITH TIME ZONE
TIMESTAMP WITH TIME ZONE
```

- A TIME WITH TIME ZONE literal has an additional time zone displacement part, consisting of an hour and a minute component preceeded by a sign, e.g.:

```
TIME '12:05:01+2:00'
```

- Time zone displacement relates local time and UTC as follows:

```
UTC time = local time – time zone displacement
```

- Thus, TIME values without explicit displacement implicitly refer to local time, i.e., an „18:00-value“ in New York and in Berlin compare to equal, even though there is 6 hours difference. Daylight savings time presents another problem!

CURRENT and LOCAL times

- For each of the three basic „datetime“ types, there are predefined value functions for „looking at the system clock“:
  - CURRENT_DATE returns the actual value of „today“
  - CURRENT_TIME returns the time at the moment („now“) of evaluation in UTC with time zone displacement +00:00.
  - CURRENT_TIMESTAMP analogously
- Each of these are parameter-less functions – optionally, a precision parameter may be added to CURRENT_TIMESTAMP for indicating length of a fractional part.
- Each occurrence of a „CURRENT expression“ within the same SQL statement is evaluated synchronously (at the same moment).
- Since SQL:1999, there are two more such value functions:
  - LOCALTIME returns „now“ in timezone-less form
  - LOCALTIMESTAMP analogously
- Each SQL session has a time zone assigned to it, which can be manipulated by the command SET TIME ZONE with options LOCAL (default) or a particular displacement value (e.g., +01:00).

Watch out for the difference in spelling! (with/without underscore)

Durations and Periods: Some General Remarks Ahead

- Periods, corresponding to mathematical intervals in 1D-space, are fundamental time objects, too. However, each period can be represented by a pair of instant denotations, corresponding to the begin and end instant of the period.

- This is the reason why there is no period data type in Standard SQL till now. Two instant valued columns have to be used for „simulating“ periods.

- The length of a period could well be measured using a single numerical data type – as are, e.g., distances, areas or volumes of spatial objects (based on the required granularity of measurement).

- However, we are used to using a variety of different temporal units if measurement in combination for expressing a duration, e.g.,

  Rather than saying „He was absent for 2045 minutes.“
  we express this fact by saying „He was absent for 1 day, 10 hours and 5 minutes.“

- Thus, we need a rather complex system of measuring units for durations, reflecting our historically motivated calendar/clock conventions, coming along with specific rules for combining them into hybrid expressions.

Hybrid Duration Expressions: Reasons for the „Month Gap“

- Hybrid duration expressions are based on (at least) six different, ordered units:
  Year – Month – Day – Hour – Minute – Second
  (which could be continued downwards by fractions of seconds).

- There is one commonly used unit of duration measurement which is missing: Week!
  Weeks and months/years are „cross-cutting“ each other, they are not combinable.

- Most units can be expressed in terms of smaller units equivalently, but not all of them!
  Unfortunately, some of the following sentences are simply wrong:

Every year has 12 months.
Every month has 28 days.
Every day has 24 hours.
Every hour has 60 minutes.
Every minute has 60 seconds.

Months may have 28, 29, 30, or 31 days.

Due to leap seconds, some very few minutes have 61 or even 62 seconds.

- The consequence is that we cannot specify the length of certain periods reasonably using month values: Whereas „1 day and 10 minutes“ is perfectly reasonable (corr. to 1459 minutes) the term „1 month and 10 days“ is ambiguous – it could correspond to either 38, or 39, or 40, or 41 days!

INTERVAL Type in SQL (1)

- Values of type INTERVAL denote durations on the (unanchored) timeline, i.e., lengths of contiguous sections between two instants – this terminology is a bit unusual (see later).

- There are two kinds of „SQL intervals“:
  - year-month intervals
  - day-time intervals.

Resist the (understandable) temptation to use plural!

- Year-month literals come in three different forms, corresponding to three variant types:
  - INTERVAL <year literal> YEAR
    where a year literal is a string containing a (possibly signed) sequence of digits, e.g., '+23' or '–5' or simply '23'
  - INTERVAL <month literal> MONTH
    where a month literal is defined similarly as a year literal
  - INTERVAL <year-to-month literal> YEAR TO MONTH
    where a year-to-month literal is a string consisting of a (possibly signed) year indicator (unconstrained, i.e., 23 years is possible), a hyphen, and a month indicator constrained to the range 0 ... 11, e.g.,

INTERVAL '2-10' YEAR TO MONTH

(i.e., two years and 10 months)

INTERVAL Type in SQL (2)

- The corresponding type names are composed of the qualifiers preceding and following the resp. value indicator, e.g., INTERVAL YEAR TO MONTH.

- Day-time literals come in various different forms, too:
  - INTERVAL <day literal> DAY
    where day literals are constructed like year literals
  - types INTERVAL HOUR, MINUTE and SECOND analogously
  - INTERVAL <day-to-second literal> DAY TO SECOND
    where a day-to-second literal is constructed in analogy to the following example: '3 2:12:35' (i.e., there is a space between the day part and the time part, itself written in colon form)
  - DAY TO HOUR, DAY TO MINUTE, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND analogously: one field per intermediate unit, e.g., '3 2:12' for day-to-minute, or '2:12:35' for hour-to-second.
  - In „range types“ like DAY TO SECOND, no field between DAY and SECOND (in this case: HOUR and MINUTE) must be omitted.

- It is forbidden to form INTERVAL literals combining units from the Year-Month group with those from the Day-Time group! Thus, an expression like this is illegal:

```
INTERVAL '1' MONTH '3' DAY
```

INTERVAL Type in SQL (3)

- Again, precision parameters can be added to the respective type names, e.g.,
  INTERVAL DAY(3)
  or INTERVAL SECOND(6,5)
  or INTERVAL DAY(3) TO SECOND(6)

- Rules for proper usage of precision parameters are:
  - For the start field in an interval type, the precision indicates the number of digits (at least 2, which is the default) for values.
  - For SECOND, the precision of the fractional part can be declared, too, e.g., SECOND(2,6). Here, 2 is the length of the leading part („before the comma“), whereas 6 is the length of the fractional part.
  - If SECOND is the trailing field, the parameter refers to the fractional part only (the integer part having default length 2).

INTERVAL: A Problematic Type Name (1)

What does „interval“ really mean?

INTERVAL: A Problematic Type Name (2)

An *interval* is a period of time, such as „3 years“ or „90 days“ or „5 minutes 30 seconds“.

(from: Date/Darwen „A Guide to The SQL Standard“)

An *interval* is an *unanchored* contiguous portion of the time line.
An interval is relative, an instant is absolute ...
The distance between two instants is an interval. Unlike instants, intervals have direction...
An interval is an unanchored, directional duration of the time line.

(from: Snodgrass „Developing Time-Oriented Database Applications in SQL“)

An *interval* is broadly defined as the difference between two dates or times.

We note in passing that SQL‘s intervals are *unanchored intervals*, meaning they describe only a duration of time without an express starting or ending time; by constrast, anchored intervals describe a specific timespan with a known start and/or finish. *Anchored intervals* are sometimes called *periods*, but that term is not defined or used in SQL:1999.

(both from: Melton/Simon „SQL:1999 Understanding Relational Language Components“)

INTERVAL: A Problematic Type Name (3)

- These quotations from various sources written by prominent researchers and practitioners are not really coherent and convincing – they show that not everywhere the „true nature“ of the interval concept of SQL is really understood the same way.

- We think that it is essential to clearly distinguish two fundamentally different concepts:
  - „Anchored portions of the time line“, as some call them, i.e., contiguous sequences of instants (of measured time, as SQL can only deal with this form of time) with an identified start and end instant, and
  - the measured length of such „portions“, i.e., durations, telling us how long some period of time lasts.

- In mathematics, an interval (from lat. „inter vallos“ = „between the posts“ of a fence) is clearly defined as the set of objects (of a totally ordered set) that are between two limiting objects – in the time context, this corresponds clearly to „portions of the time line“, also called periods.

- A duration is clearly a different thing than a period (in the sense just mentioned) – it is the length of a time interval (in the mathematical sense), not the interval itself, thus:

The choice of the term *INTERVAL* in SQL is extremely misleading – be very careful!

„Arithmetics“ for Time (1)

- The usual arithmetic operators (+, –, *, /) are applicable to time values, too, but come in various different forms underlying various restrictions – treat them very carefully!

- A first form of addition/subtraction is intended to „move“ instants on the time line in forward or backward direction. An interval value can be added/subtracted from a datetime value (not vice versa), e.g.:

```
TIME '12:45:00' + INTERVAL '90' MINUTE
CURRENT_TIMESTAMP – INTERVAL '1' DAY
```

- Of course, the interval variant is expected to „fit“ with the datetime operand!

- In this form, +/- always results in another datetime value (of the same variant/granularity as that of the datetime operand).

- Datetime values can be subtracted from each other (not added!), too, resulting in an interval value indicating „how far“ they are apart from each other on the time line, e.g.:

```
TIME '14:15:00' – TIME '12:45:00'
```

(returning INTERVAL '1:30:00' HOUR TO SECOND)

„Arithmetics“ for Time (2)

- Intervals may be added/subtracted, resulting in other intervals. e.g.:

```
(TIME '14:19:00' – TIME '12:45:00') + INTERVAL '90' MINUTE
```

- If doing so, don‘t „mix“ year-month and day-time intervals!

- However, the following is acceptable

```
DATE '1999-12-01' + INTERVAL '1' MONTH + INTERVAL '1' DAY
```

  if evaluated in this order:

```
( DATE '1999-12-01' + INTERVAL '1' MONTH ) + INTERVAL '1' DAY
```

- Finally, an interval can be multiplied/divided with/by a number, resulting in an interval, the length of which is a multiple/fraction of the operand interval, e.g.:

```
INTERVAL '2' DAY * 3
INTERVAL '6' DAY / 2
```

„Arithmetics“ for Time (3)

This table summarizes all the combinations of operands, operators, and results that are permitted in SQL:

1st operand	operator	2nd operand	result
datetime	+	interval	datetime
datetime	–	interval	datetime
interval	+	datetime	datetime
datetime	–	datetime	interval
interval	+	interval	interval
interval	–	interval	interval
interval	/	number	interval
number	*	interval	interval

Type Conversion and Casting – and EXTRACT

- Last not least, there are various ways how to convert temporal values to values of other data types, and vice versa – using the CAST .. TO .. function. We don‘t cover all variants here, but just indicate the idea using a few examples:

```
CAST '10:00:00' TO TIME
CAST 3 TO INTERVAL YEAR
CAST INTERVAL '7' MINUTE TO NUMBER
```

string to time
number to interval
interval to number

- Another useful function for manipulating temporal values is EXTRACT ... FROM ..., suitable for extracting components from individual datetime values (returning a number), e.g.:

```
EXTRACT YEAR FROM '2010-01-01'
```

(other options are MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, TIMEZONE_MINUTE)

Theory and Practice: Standard Conformance of DBMS Products

3.5 IMPLEMENTATION CONSIDERATIONS

Although temporal types have been in the SQL standard since 1992 and were defined in the mid-1980s, it is surprising, and unfortunate, that unlike other portions of SQL, the types and their predicates and constructors are not supported by most DBMSs. Instead, each vendor has defined an incompatible and idiosyncratic set of temporal types and operators, replete with inconsistencies and seemingly arbitrary design decisions. Temporal types are among the most variable features of commercial DBMSs. Coupled with this is the often poor documentation available from the vendors of temporal features of their products. Determining the operations supported on temporal type(s) can be a frustrating exercise, with the information, if present at all, spread across the documentation. The following is an attempt to gather in one place the information about temporal support in a few prominent DBMSs.

No vendor supports SQL-92 at the Full SQL level of conformance. All products include idiosyncrasies in their temporal support that render porting to other DBMSs difficult.

(from: Snodgrass‘ book, p. 42)

Don‘t expect that current DBMS products support this part of the standard at all!
Expect a lot of proprietary styles quite far from the clean standard concept instead!
Consult chapter 3.5 in the Snodgrass book for details – but look at vendor info, too.

PERIOD: The Temporal Data Type Missing in SQL

- Up till now, there is no datatype for directly expressing anchored intervals (aka periods) in Standard SQL (and thus in most of the vendor dialects).
- Teradata SQL seems to be the only DBMS product supporting a PERIOD type till now.
- This is quite surprising, as many research proposals have been made wrt such a type.
- The reason for excluding periods up till now has (probably) been that each period can be „simulated“ by a pair of columns of the same instant type (*From*, *To*).
- The consequence of this decision is the lack of operators for directly manipulating or comparing periods, which is leading to rather complex SQL queries if several periods are involved.
- There is a (rather ugly) „compromise operator“ though, OVERLAPS, which tries to overcome the missing datatype „through the backdoor“ – see below.
- We already discuss certain basic properties of periods here, but postpone ideas for an extension of the SQL standard to a later chapter (when discussing perspectives in general).

Why Periods?

- The main motivation for proposing inclusion of a PERIOD data type comes from the frequent need to assign periods to objects (and to store this information in a single fact).

- On the one hand, there are periods appearing in real life applications that have to be recorded (and cannot be predicted or computed), e.g.,
    - contract periods
    - absence from the job due to holidays or illness
    - loaning objects (e.g., books from a library)
    - term for finishing one‘s master thesis

- On the other hand, we will see that it might be necessary to assign periods to any fact in a database automatically in order to record how long the resp. fact has been stored in the DB in this form (history databases, logs).

- Of course, periods can be simulated by means of pairs of instant columns (representing begin and end of the period, resp.). However, the overhead incurred by expressing operators and predicates on periods in terms of operators and predicates on pairs of instants is severe and often intolerable.

Close-Open v. Close-Close Periods

- In our German university system, each winter semester starts on October 1st, whereas each summer semester starts on April 1st.

- Thus, there are two practically usable alternatives for denoting the periods covered by, e.g., the previous winter semester:
  - By means of an interval closed on both ends: [1.10.2010, 31.3.2011]
  - By means of a „half-open“ interval, closed only at the start: [1.10.2010, 1.4.2011)

  Which one to prefer over the other (and why)?

- In theory, there are three more alternatives: open-close, open-open, and start-duration, but all three are not really useful in practice.

- There are but few convincing arguments pro/con one of the two realistic alternatives – many authors prefer the close-open style, e.g., because it avoids problems that arise if moving to lower granularities: [1.10.2010 0:00, 31.3.2011 23:59] looks akward. If choosing 1.4.2011 as end date in a closed interval style, adjacent periods would overlap.

OVERLAPS (1)

- SQL does not support „real intervals“ anchored on the time line, which we called periods previously (and have been called like that in SQL extension proposals).

- Such periods have to be „simulated“ by using two datetime columns for start and end instant of the resp. „portion of the time line“.

- However, SQL (as of today) „opens up to the world of periods“ by offering a predicate OVERLAPS for testing whether pairs of datetimes – interpreted as start/end of a period – do have at least one instant in common, e.g.:

two variants for expressing „period constructors“

```
(TIME '12:45:00' , TIME '14:15:00')
        OVERLAPS
(TIME '13:30:00' , TIME '15:00:00')
```

evaluates to true

```
(TIME '12:45:00' , INTERVAL '1' HOUR)
        OVERLAPS
(TIME '13:50:00' , TIME '15:10:00')
```

evaluates to false

- Open-ended periods are possible, too, leaving the end NULL.

OVERLAPS (2)

- Despite the style of writing, using round brackets on both sides, the two arguments of of any OVERLAPS expression are interpreted as [close, open) intervals on the time line!
- The syntax of OVERLAPS has been defined in obvious ignorance of the mathematical tradition of denoting intervals – which is not surprising after the „ugly“ usage of the term INTERVAL for durations.
- As a consequence of the intended closed begin, there is no NULL in the past!
- A further consequence of this choice is that, e.g.,

```
(TIME '08:00:00' , TIME '09:00:00')
OVERLAPS
(TIME '09:00:00' , TIME '09:30:00')
```

  evaluates to false!
- Another implicit assumption (in line with the [close, open) interpretation, is that the start instant is always earlier (smaller) than the end instant.
- Thus, an expression like

```
(TIME '09:00:00' , TIME '09:00:00')
```

  is simply undefined (not legal) in combination with OVERLAPS.

OVERLAPS (3)

- The SQL semantics of OVERLAPS is a bit tricky – we will learn about a different way of defining this predicate in a moment:

(L^{start}, L^{end}) OVERLAPS (R^{start}, R^{end})

evaluates to true iff

$(L^{start} > R^{start}$ AND $(L^{start} < R^{end}$ OR $L^{end} < R^{end}))$
OR
$(R^{start} > L^{start}$ AND $(R^{start} < L^{end}$ OR $R^{end} < L^{end}))$
OR
$(L^{start} = R^{start}$ AND L^{end} IS NOT NULL AND R^{end} IS NOT NULL$))$

- Fortunately, there is a very easy intuition behind this „formal monster“:

(L^{start}, L^{end}) OVERLAPS (R^{start}, R^{end})
is true if and only if the two periods
$[L^{start}, L^{end})$ and $[R^{start}, R^{end})$
have at least one instant in common!

SQL OVERLAPS Reconsidered

SQL OVERLAPS operator problem, how to get rid of it

1

I expect that date period from '2011-01-28' to '2011-02-01' OVERLAPS period from '2011-02-01' to '2011-02-01' (that's the same day here), but it does not!

PostgreSQL expecting the match of exact ending point is NOT a match...how to get rid of this? I would like to have it treat the above scenario as overlap.

```
SELECT (DATE '2011-01-28', DATE '2011-02-01') OVERLAPS
       (DATE '2011-02-01', DATE '2011-02-01');
```

returns false, while I expect it to return true.

from:
http://stackoverflow.com/questions/52183700/sql-overlaps-operator-problem-how-to-get-rid-of-it

Not possible according to the standard!
INTERVAL literal expected!

In order to not confuse everybody about the inconsequent style of representing periods in SQL, we will use the strict [close,open) style **throughout this lecture**!

4

You expect wrong. From the fine manual:

> Each time period is considered to represent the half-open interval *start* <= *time* < *end*, unless *start* and *end* are equal in which case it represents that single time instant. This means for instance that two time periods with only an endpoint in common do not overlap.

So, if you want the closed interval, *start* <= *time* <= *end*, then you can either do end-point checks explicitly as Catcall suggests or you can add a single day to the upper bound:

```
SELECT (DATE '2011-01-28', DATE '2011-02-01' + 1) OVERLAPS
       (DATE '2011-02-01', DATE '2011-02-01'    )
```

But be careful to put the end-points in the correct order as:

> When a pair of values is provided, either the start or the end can be written first; OVERLAPS automatically takes the earlier value of the pair as the start.

OVERLAPS „Periods“ (2)

This means that we will have to write overlap of a „proper“ period with exactly one timepoint (here: one day) as follows:

```
(CURRENT_DATE, CURRENT_DATE + INTERVAL 1 DAY)
        OVERLAPS
(H.START_DATE, H.END_DATE)
```

Of course, this is a bit tedious and even counterintuitive – that‘s why the SQL designers introduced the exception for single-instant „periods“, which allows you to write:

```
(CURRENT_DATE, CURRENT_DATE)
        OVERLAPS
(H.START_DATE, H.END_DATE)
```

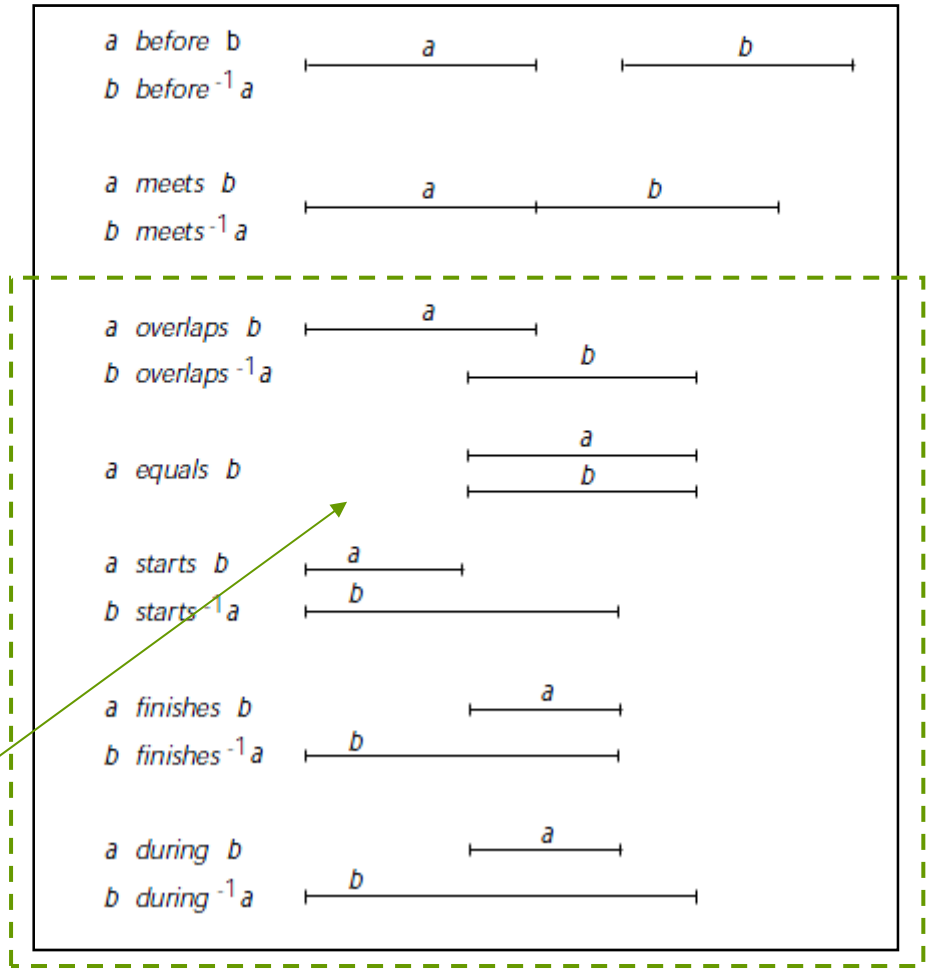
I believe that it is worthwhile to accept this extra effort (+ 1) in order to stick to a clean and unique style of using the same kind of interval notation throughout.

„Allen Operators“ for Comparing Intervals in General

If looking at intervals of any kind (not only periods), one can identify 7 different ways (13, if including inverses) how intervals in general (and periods in particular) can be positioned relative to each other:

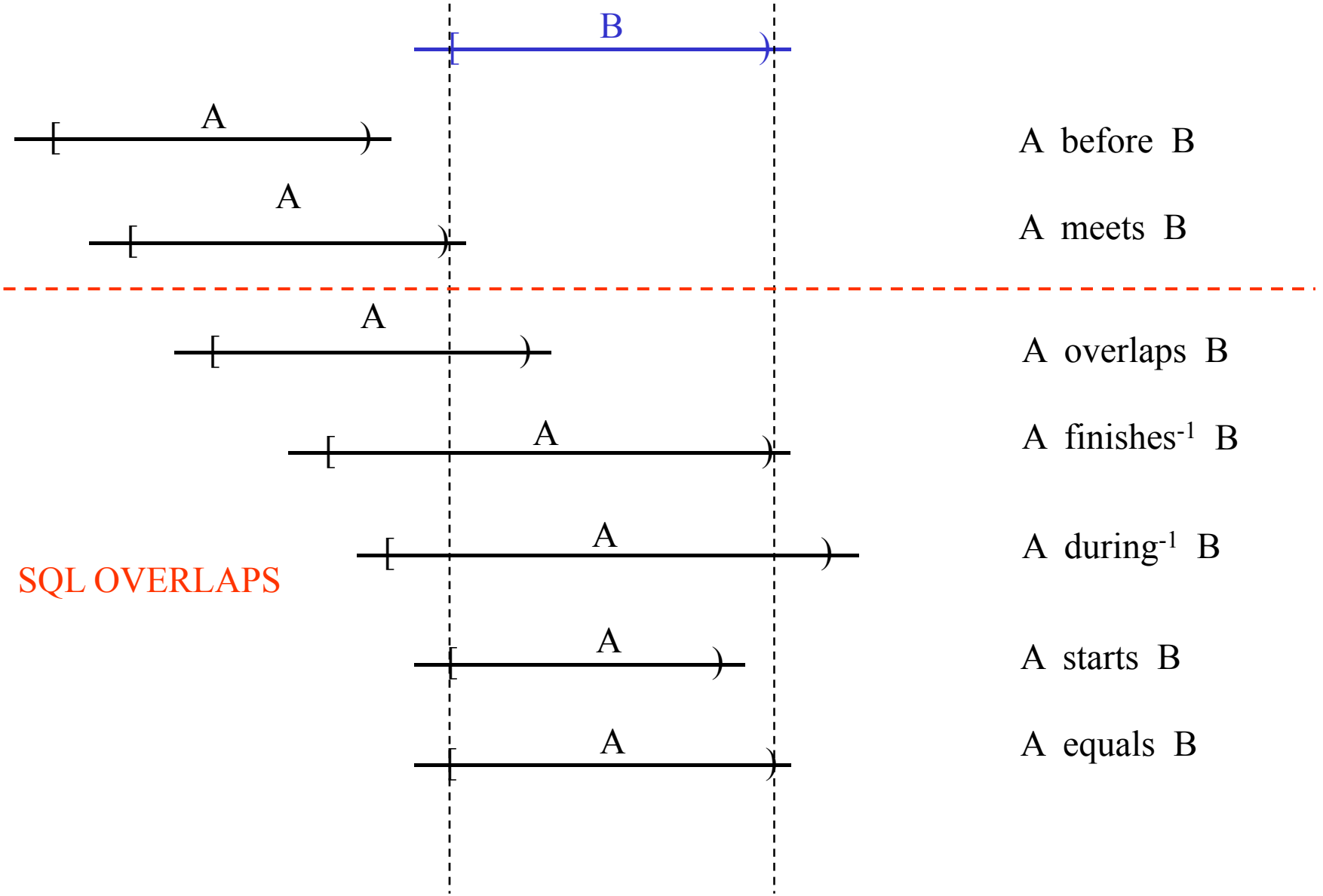
The corresponding Boolean operators have been named and first proposed in a famous paper by James Allen in 1983 (and are thus often called „The Allen Operators“):

„Maintaining Knowledge About Temporal Intervals“, CACM Vol. 26 (11)

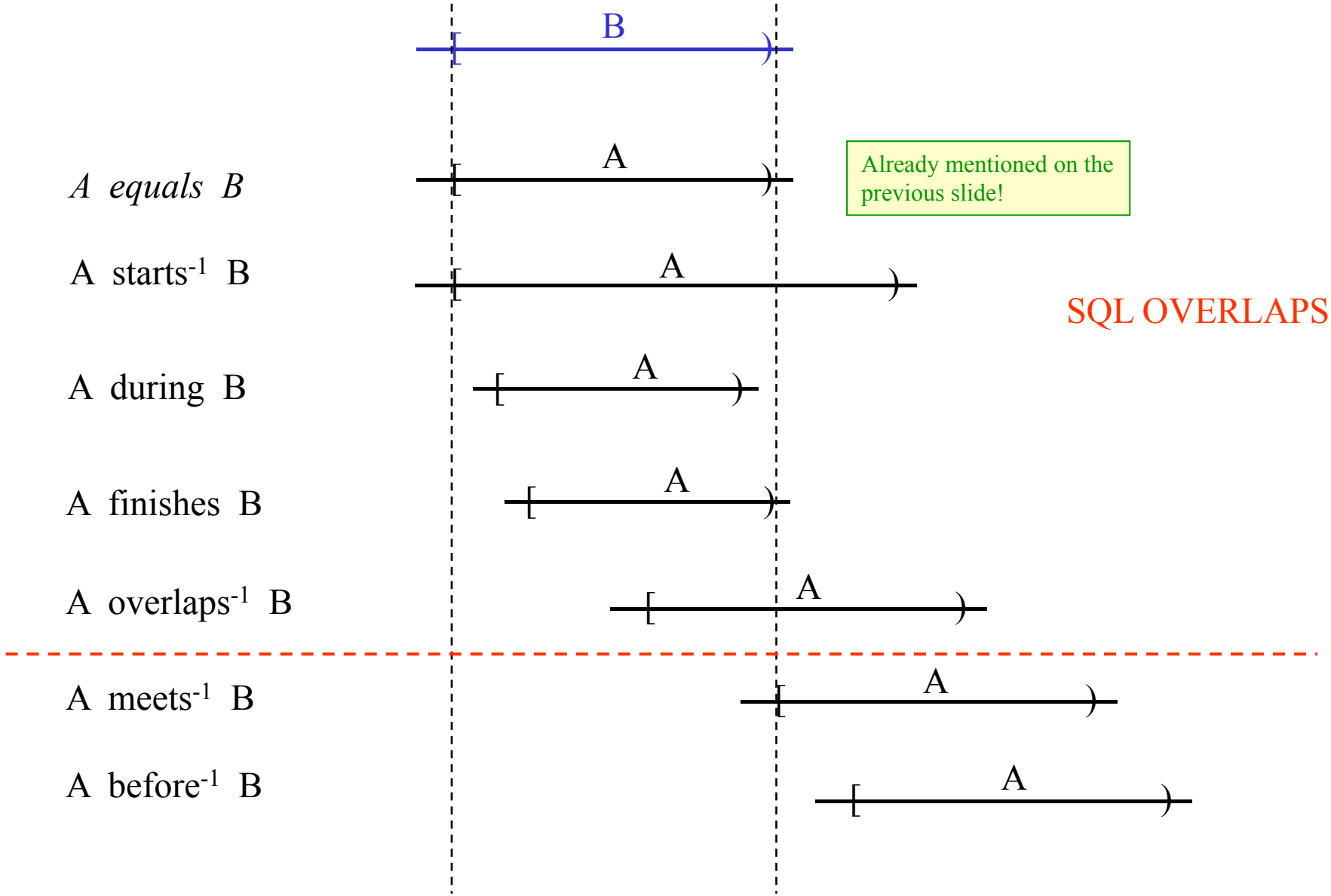


SQL OVERLAPS covers no less than 9 of these operators by means of just one!

Allen‘s Relationships Systematically Arranged (1)



Allen‘s Relationships Systematically Arranged (2)



Perspectives

- For now, we will stick to the current state of affairs in the „SQL World“, i.e., we will do with just the „simulation“ of periods offered today.
- Research perspectives as well as very recent developments (SQL:2011) will be left for a later chapter (Chapter 5).
- In the two following chapters, we will try to cope with what is available today in the commercially available DBMS products, mostly supporting old standard versions.
- As both styles of dealing with temporal data („Time about Data“, „Data about Time“) will be based on period timestamping, this drawback will have rather severe consequences, though.
- But even if you will have a proper period type supported by standard and/or products sooner or later, you will have to understand what its new operators mean in terms of the complex (*From*, *To*) style.