Temporal Information Systems

SS 2018

Representing Time in SQL

Chapter 2

Temporal Objects and Types in SQL: A Quick Reminder

- 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

- In „classical“ SQL there are data types for instants and durations (of periods) only:
  - for instants: DATE, TIME, TIMESTAMP
  - for durations: INTERVAL

- Periods are not regarded as „first class“ temporal objects in SQL, but have to be simulated by means of pairs of instant-typed columns (of the same type).

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?

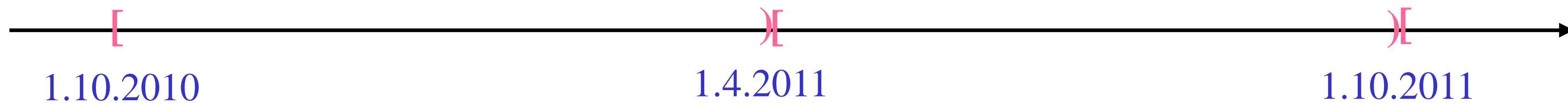
- The main motivation for proposing inclusion of a PERIOD data type comes from the frequent need to assign periods to objects (and to store this information in a single fact).

- 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.

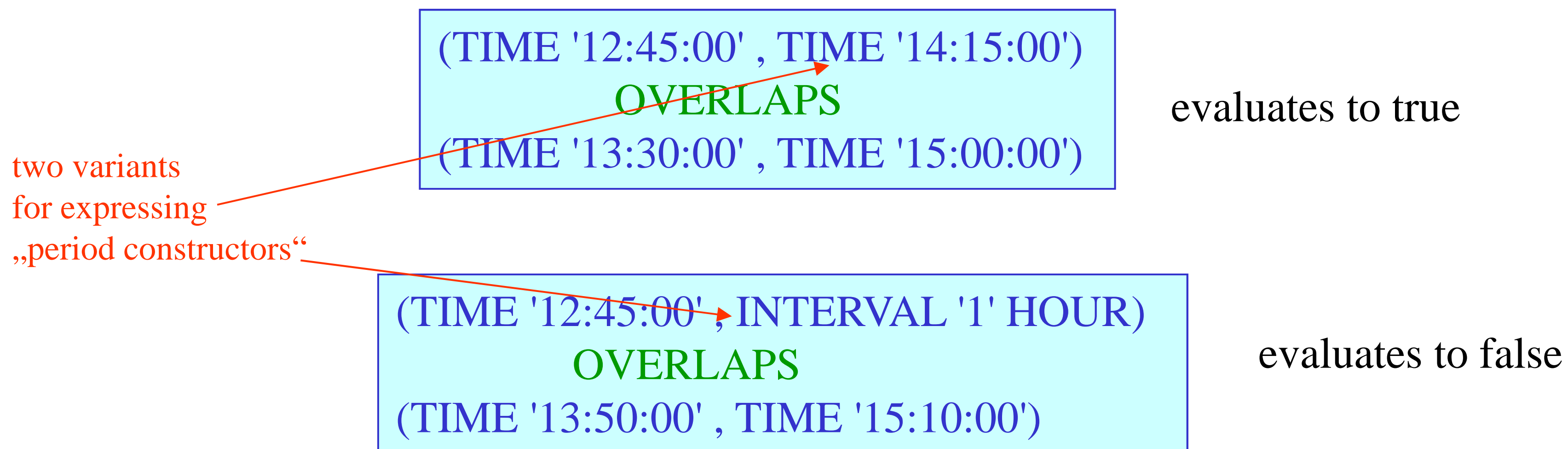
- There are but few convincing arguments pro/con one of the two realistic alternatives – many authors prefer the close-open style, e.g., because it avoids problems that arise if moving to lower granularities: [1.10.2010 0:00, 31.3.2011 23:59] looks akward. If choosing 1.4.2011 as end date in a closed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



- 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

CAREERS 2.0 by stackoverflow

+

Have projects on Codeplex?
Import them easily to your profile

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!

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.

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**!

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)
```

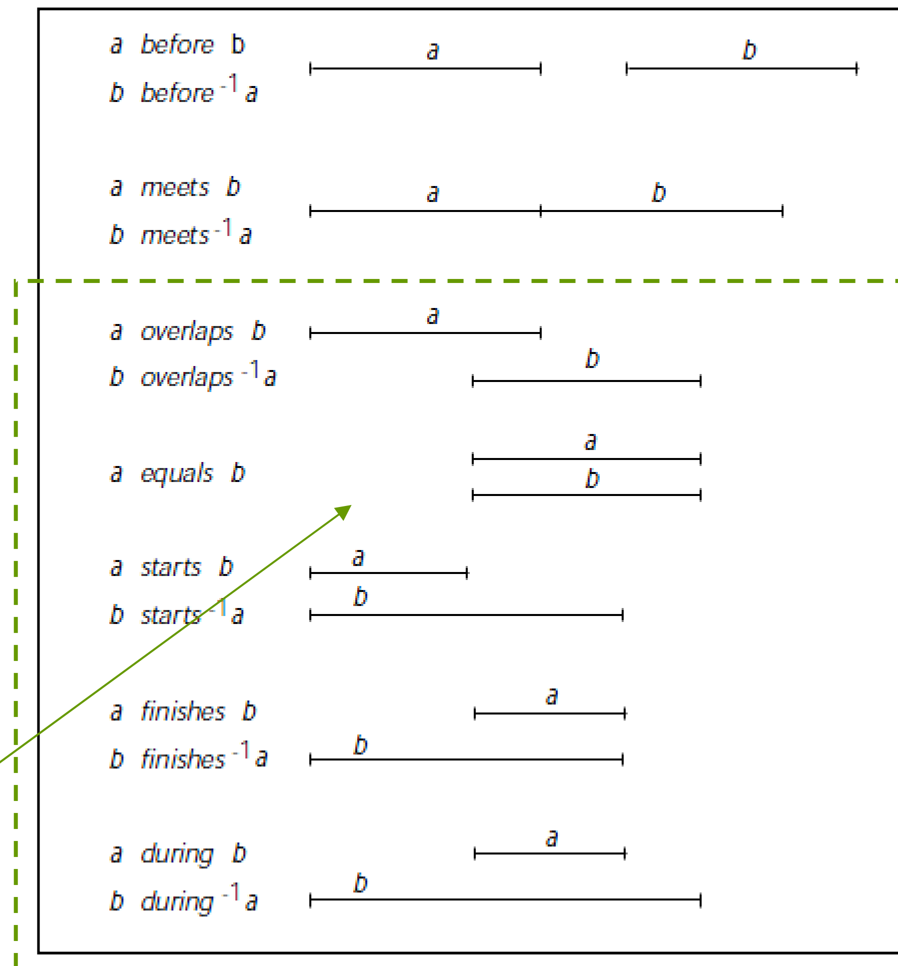
I believe that it is worthwhile to accept this extra effort (+ 1) in order to stick to a clean and unique style of using the same kind of interval notation throughout.

„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:

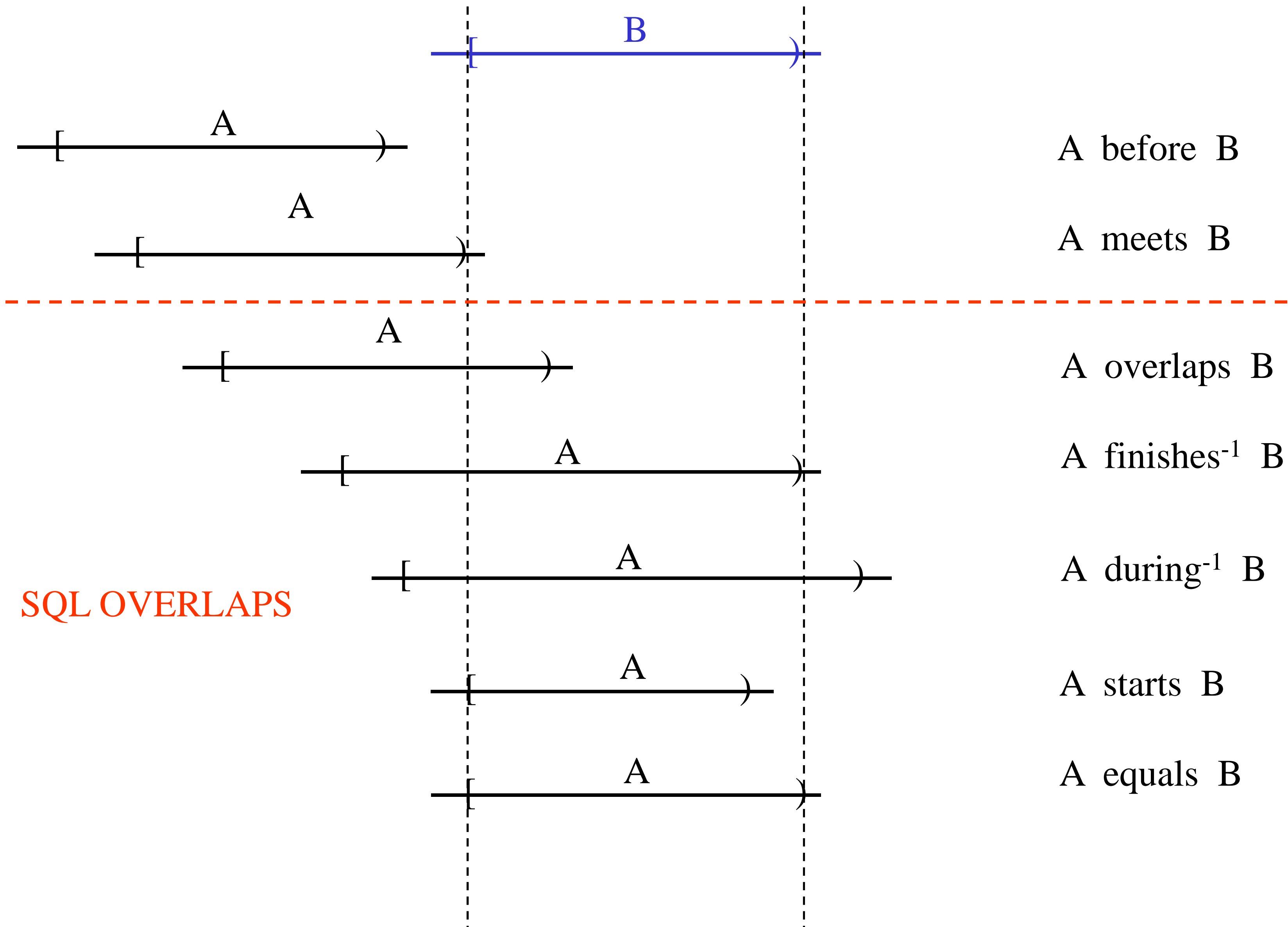
The corresponding Boolean operators have been named and first proposed in a famous paper by James Allen in 1983 (and are thus often called „The Allen Operators“):

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

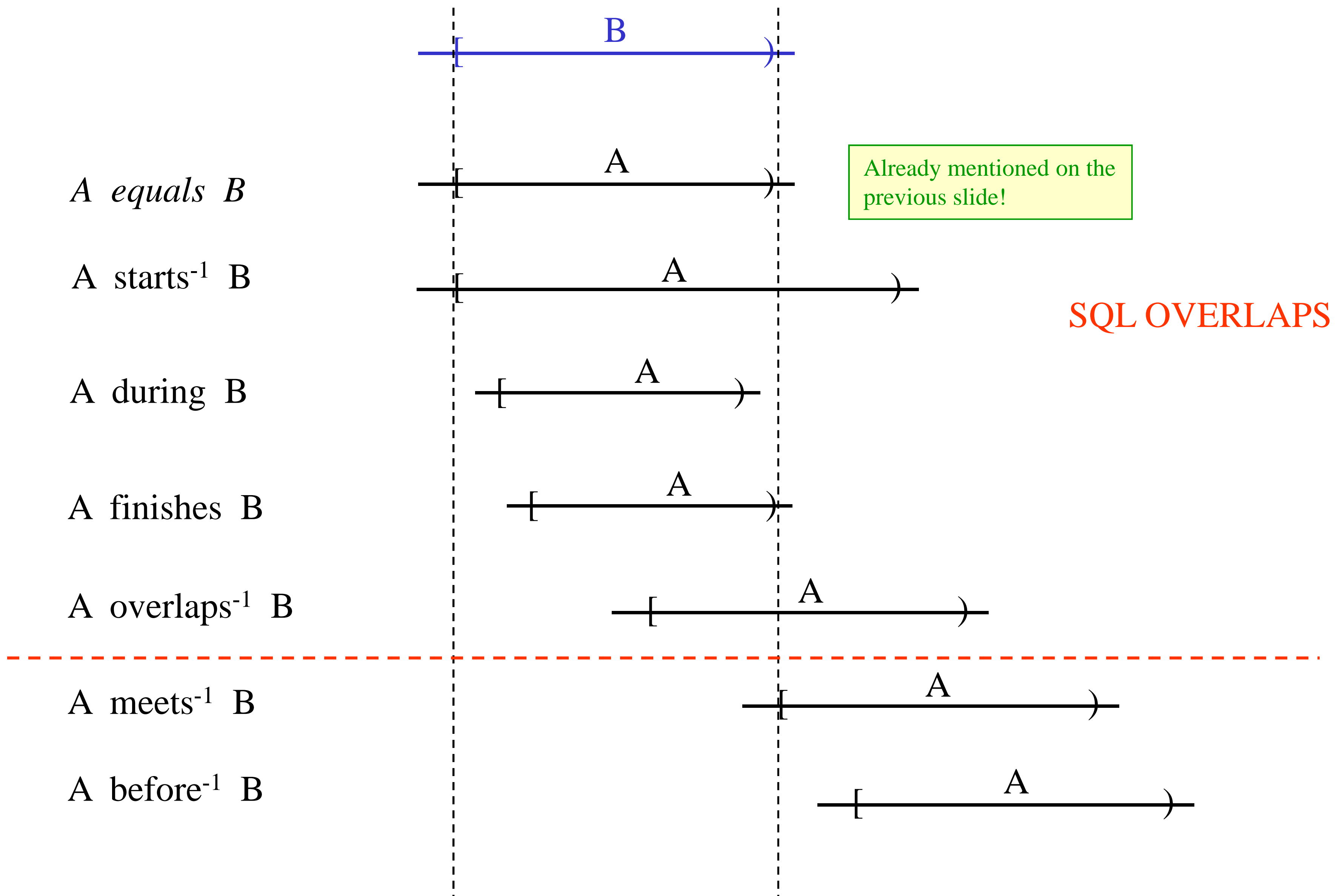


SQL OVERLAPS covers no less than 9 of these operators by means of just one!

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.

SS 2018

„Time About Data" – Keeping a History of Change

Chapter 3

IS and Change: Some Philosophy Ahead (1)

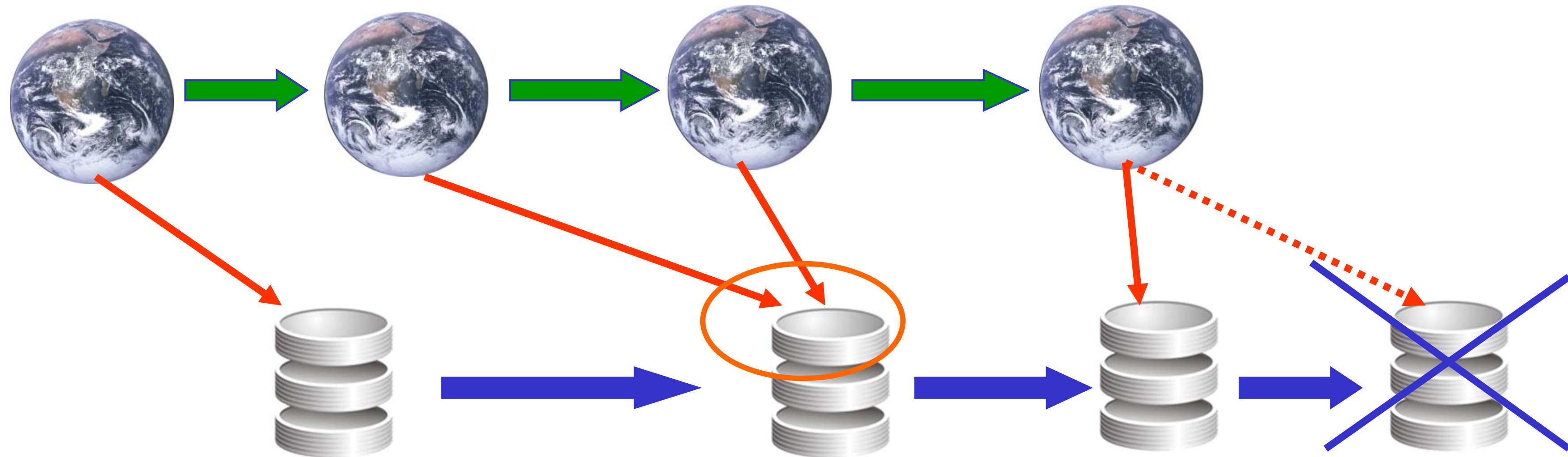
Prevailing philosophy of information system design and usage:

At every instant of time:

The contents of the information system reflects a particular state of the „world“!

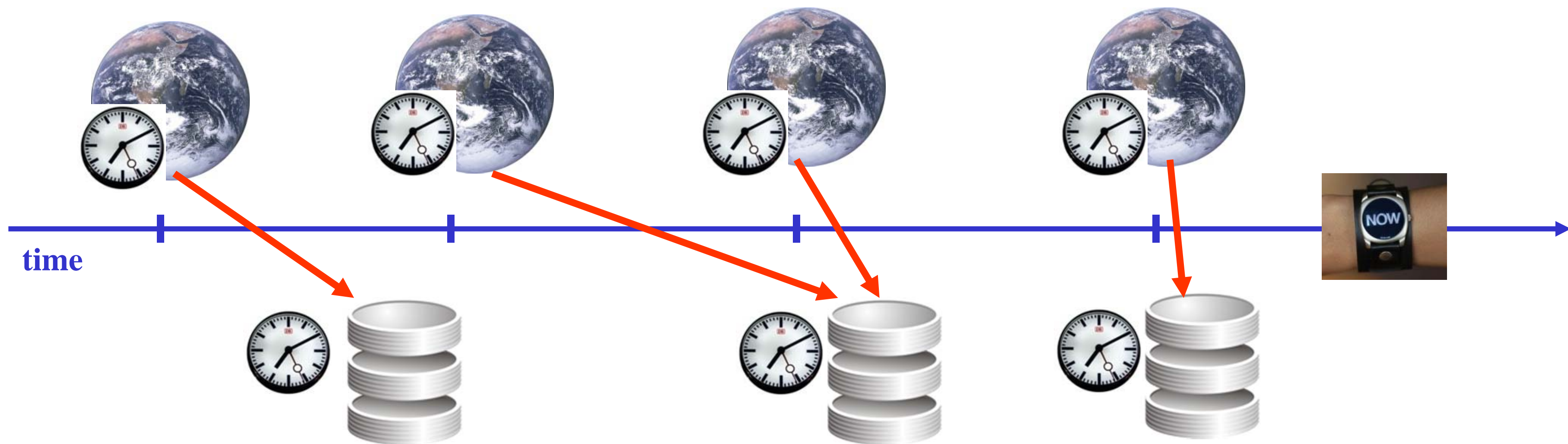
Databases represent snapshots of the world (ideally, as it is just now)!

IS and Change: Some Philosophy Ahead (2)



- Changes of „the world“ ought to be reflected by changes of the IS (we can‘t be sure, though!).
- IS evolution is always delayed wrt world evolution, but it often changes synchronously.
- However, there may be cases where the world changes more often than the IS:
  Several changes of the world may be reflected by a single change of the IS.
- An IS evolves only if the world has changed before –
  IS never change without a cause originating from „the world“.

IS and Change: Some Philosophy Ahead (3)



- Every change in the world takes place at a particular moment in time. So does every change of the database!
- This moment in time might be unknown, or considered irrelevant for the IS.
- Thus, the time of change is not always recorded in the IS.
- Even the sequence of changes occurring might be lost once reflected in the IS.
- Many (most?) IS do just represent the current state of the world, but no history at all!

© 2018 Prof. Dr. R. Manthey

Databases That Never Forget

> „The database is not the database – the log is the database, and the database is just an optimized access path to the most recent version of the log“.

(B.-M. Schueler in „Update Reconsidered“, 1977)

- This chapter will be concerned with techniques of keeping track of all changes of certain tables of a temporal database by logging each change and keeping all versions of the resp. tables.

- Such versioned databases are required nowadays in a wide variety of application domains, in particular when legal problems (e.g., liability and auditing) have to be expected. More and more often, the provenance (history of origin) of data has to be proved.

- The technical key decision for such services is to automatically record all changes without human users being able to influence this process (administrators included). Key problem is how to properly query such databases (and, sometimes, how to update them properly).

Reminder (From Chapter 0): Logs

Log: originally „part of the stem of a tree"

(The „log lady" from the TV series „Twin Peaks")

Later: „Tally stick", used for recording consumption

Even later:
Ship's logbook, used for recording changes of direction and relevant events of a trip

Another Reminder: Log and Logging in Transaction Management

- The terms log/logging are well-known in the DB context, as they have been around in the context of transaction management in a DBMS since many decades.

- Transactions are sequences of change statements treated as a unit, i.e., they are either performed entirely (and successfully), or not at all, not even partially, in case of „failure“ of at least one of the component operations (atomicity of a transaction).

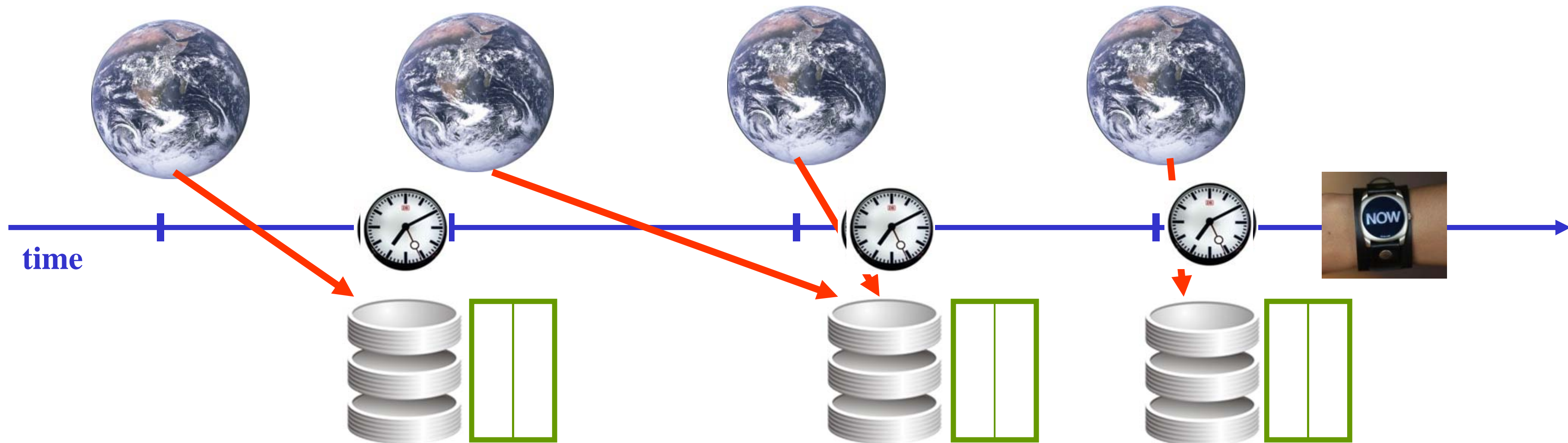
- Every DBMS controls every transaction wrt physical and logical consistency and protects and controls execution order in case of „competing“ transactions simultaneously trying to modify the same data (synchronisation).

- If any unresolvable problem occurs, execution of the affected transaction is stopped, and all changes to the DB already performed are rolled back till a consistent previous state has been reached (recovery).

- In order to be able to perform rollback, a temporary log of all performed operations of each active transaction is kept by the transaction manager of the DBMS.

The log we are speaking about here, is a different one – kept permanently!!

Keeping Data About History



- In this chapter, we will clearly separate the data part of a relational tuple from the history part of that same tuple:

  Timestamps added to tuples (representing facts in reality) denote those periods during which the resp. tuple was current in the database.

- Current changes in reality (called logical modifications) are always translated into physical modifications of the history DB preserving past data and recording the time of modification by means of start/end timestamps.

- Without explicitly mentioning, we assume that timestamps refer to the time of DB modification rather than to the time of „change in the real world“.

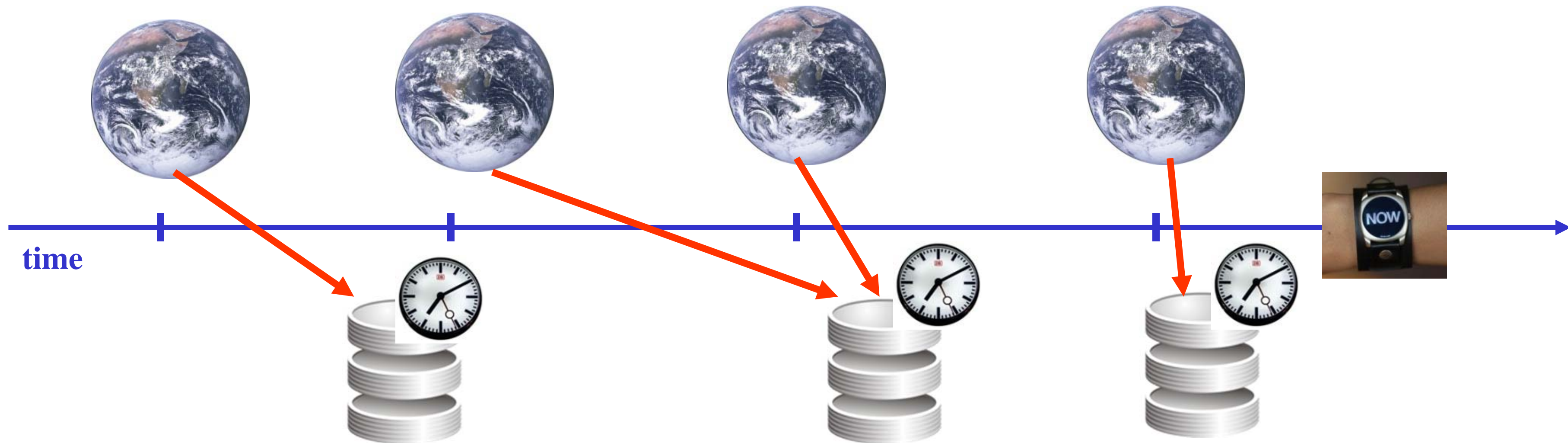
Timestamp: Dual Meaning – Beware of the Ambiguity!

The term „timestamp“ has been used on the previous slide in a different sense than in chapter 2 – both forms of usage are common in TDB research!

1) In SQL, there is the data type TIMESTAMP consisting of DATE-TIME values.

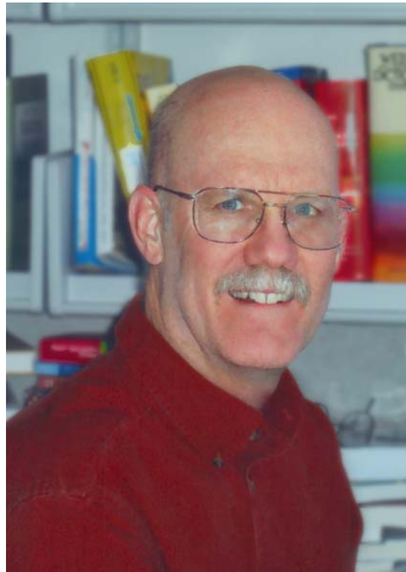
2) In databases keeping history of change of tables, the value of some temporal data type added to each tuple (representing facts in reality) in order to indicate when these tuples were „valid“ in reality or in the database, the additional temporal value is called the timestamp of the resp. tuple.

Transaction Time



- In temporal DB research, storing those instants when data change in the DB, resp., those periods when data were current in the DB, is called keeping transaction time.
- Ideally, transaction timestamps are referring to the system clock, not to the watches of humans issuing modification commands. Again ideally, transaction time timestamps are generated automatically by the DBMS.
- Even more ideally, TT timestamps cannot be modified by human users later on anymore!
- In comparison, time of change in reality will be called valid time.

ATTENTION: We are Using Transaction Time for History Keeping!



- This chapter mainly follows chapters 5-7 in the book by Snodgrass, but . . .
- . . . whereas Snodgrass discusses history keeping in terms of timestamps referring to the clock of „the world“ (valid time),
- we do so in terms of the clock of the DBMS (transaction time)! We will look at valid time later.