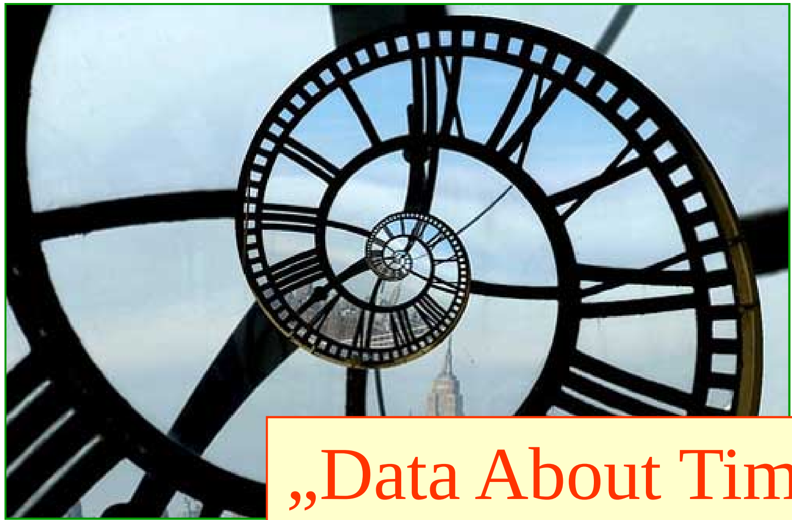
Temporal Information Systems

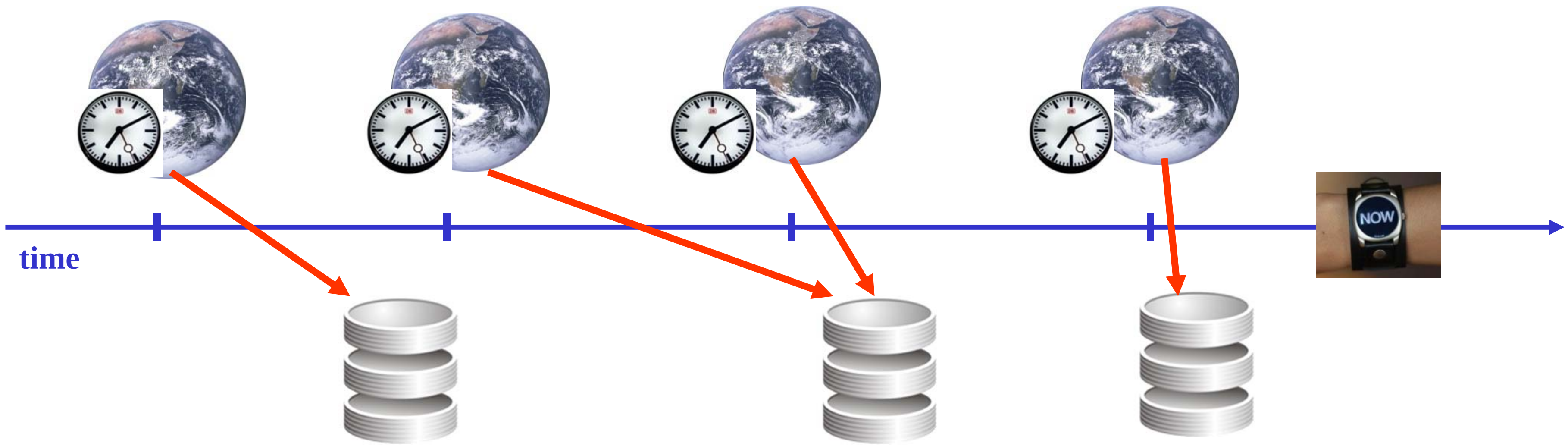
SS 2015



„Data About Time“ – Managing a History of the Application

Chapter 4

Valid Time



- In this chapter, we turn our attention to time-related information which
  - is referring to events occuring and validity of facts in that part of the real world reflected in the resp. database,
  - is therefore kept in columns in the data part of each tuple (not in the history part),
  - is thus inserted/modified by humans (or programs) „monitoring“ the resp. part of the real world (not by „the system“, i.e., the DBMS).

- Columns of tables containing elements of temporal data types which refer to the real world outside the DB are called in research valid time columns.

Timestamping (1)

- The term „timestamping“ has been used just intuitively up till now and requires some clarification.

- Up till now, a „timestamp“ always was a period value „attached to“ an entire fact in order to record its period of validity in the database, e.g.:

Student	Class	Signed_up	Dropped	Grade	Exam Date	From	To
John	1203	11.11.2010				11.11.2010	14.2.2011

timestamped fact — timestamp of that fact

- Here, the „object“ to be timestamped is a fact, the timestamp itself is a period value, and the temporal status of the timestamp is transaction time:

  We have a case of transaction time tuple timestamping with period granularity.

- However, there are other forms of timestamping imaginable, using
  - valid time as status of the timestamp (rather than transaction time)
  - instant granularity for the timestamp (rather than period)
  - timestamping individual columns only (rather than the entire fact)

Timestamping (2)

- In the presidency table, two forms of timestamping can be observed simultaneously:
  - The tuple recording the first US presidency ever (covering columns *Presidency*, *President*, *Term*) is timestamped according to its occurrence in the real world:

Presidency	President	Birthday	From	To	Term
1	George Washington	22.2.1732	30.4.1789	4.3.1793	1

timestamped tuple

valid time period tuple timestamp

  - The *Birthday* column could be interpreted as a valid time timestamp for the value in column *President* – although doing so „stretches" the idea of time-stamping quite a bit:

Presidency	President	Birthday	From	To	Term
1	George Washington	22.2.1732	30.4.1789	4.3.1793	1

timestamped attribute value

valid time instant attribute timestamp

Timestamping (3)

- In the bi-temporal table *Exams* the two valid time columns *Signed_up* and *Dropped* can be interpreted as valid time period timestamp for the facts consisting of columns *Student* and *Class* stating who has registered for which class.

- If considering *(Student, Class, Grade)* as separate tuples recording which student took an exam in which class, then *Exam Date* can be interpreted as a valid time tuple time-stamp. Interpreting it as an attribute timestamp for attribute *Grade* is possible, too – this seems to be particularly useful if two exams are possible per class.

Student	Class	Signed_up	Dropped	Grade	Exam Date	From	To
John	1203	11.11.2010				11.11.2010	14.2.2011
John	1203	11.11.2010		1,3	13.2.2011	14.2.2011	
Jack	1203	19.11.2010				19.11.2010	2.1.2011
Jack	1203	19.11.2010	2.1.2011			2.1.2011	
Tim	1203	21.11.2010				21.11.2010	20.3.2011
Tim	1203	21.11.2010		3,0	18.3.2011	20.3.2011	8.4.2011

valid time timestamps (one period, one instant)

transaction time period tuple timestamp

Events vs. States

- Instant timestamps correspond to events happening which are associated with the timestamped object (i.e., fact or value) in some sense (e.g., „moment of creation“). In this interpretation, events do not have duration, but happen instantaneously.

- Period timestamps are associated with timestamped objects in order to record how long these objects have been in a particular state. Thus states of objects have duration – all attributes of the object (which are recorded) are stable (not changed) while the object is in the resp. state.

- When an object changes, a new state of that object is created. State changes are events, delimiting the period during which the object is in that particular state. Thus, recording just change events or full periods are two options for representing stateful objects.

- In natural language or in philosophy (and other branches of science) there is no common agreement on the question, whether events can have duration, too (or are „by nature“ instantaneous) and whether states can be instantaneouos (or have duration „by nature“).

- Last not least: Not every temporal column of a table must be a timestamp!

On Events

„We think that the most important distinction among methods of managing queryable data is the distinction between data about things and data about events.

Things are what *exists*; events are what *happen*.

Things are what change; events are the occasions on which they change.“

(from Johnston/Weis „Managing Time ...“, p. 37)

„Events are the occasions on which changes happen to persisting objects.
As events, they have two important features:
(i) they occur at a point in time, or sometimes last for a limited period of time;
and (ii) in either case, they do not change.
An event happens, and then it‘s over. Once it‘s over, that‘s it; it is frozen in time.“

(from Johnston/Weis „Managing Time ...“, p. 37)

State vs. Event Tables

- In case of tuple timestamping, the type of the timestamp for each tuple decides whether we keep history about this tuple in terms of states, or of events.

- If periods are used as tuple timestamps, we call the respective table a state table, if instants are used we speak of an event table.

- The attribute *state* vs. *event* has to be further qualified by the time dimension to which it applies, e.g., valid time (VT) state table, or transaction time (TT) event table.

- A bitemporal table can be a valid time event table and simultaneously a transaction time state table. All four combinations are possible analogously.

- The most frequent form of usage of timestamping is the state table style, i.e., recording periods of validity of the recorded fact in reality (VT), resp. periods of unchanged con-tainment of the recorded tuple in the database (TT).

- An important special case of a state table is called a snapshot table. Here, all period time-stamps are „degenerate“ in that they represent instants (periods of duration 1), and all tuples have the same timestamp: What was true (resp., known) at that instant?

Managing Valid Time State Tables: Principles

- In this (short) chapter, we will first look at those aspects of data management that are different if dealing with valid rather than transaction time.
- For the rest of the chapter, we will look at **state** tables only (as in chapter 3 before), but this time pairs of columns representing periods will be interpreted as valid time timestamps.
- Again, we will distinguish the data part of a row from its history part. The history part of a row will refer to periods in the application domain of the resp. database, however.
- After discussing VT-specific issues from the perspective of „old“ SQL (using the terminology of temporal DB research), we will again turn to SQL:2011 and introduce the novel syntactic features of the latest standard.
- Querying a valid time table works like querying a transaction time table, unless using SQL:2011, of course. If using „ordinary“ SQL, no difference between temporal and non-temporal columns exists wrt querying.
- Current modifications are treated similarly to the TT case. However, for VT tables (past and) sequenced modifications become meaningful and have to be discussed.

Sequenced Insertions

- Information about events and states in the application area represented by the data in the DB are relying on communication with the „real world“. Humans have to take care of „translating“ the contents of such communication to the DB. Information about past events in the application world may thus be erroneous or (strongly) delayed!

- Sequenced insertions are physically realized similar to current insertions, e.g.:

```
INSERT  INTO INCUMBENTS
VALUES (111223333, 999071, DATE '1997-01-01', DATE '1998-01-01')
```

- Note that such an insertion will be applicable only if there is no other assignment of this position to this employee during any instant of the respective period, if a temporal primary key is active on INCUMBENTS.

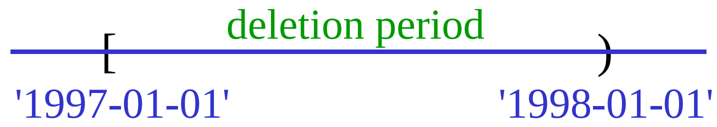
- Past insertions are treated similarly with the period „degenerating“ to an instant.

- Next let us try to express a non-temporal (logical) deletion for the entire year 1997 in retrospect, i.e., turn it into a sequenced deletion.

Sequenced Deletions (1)

```
DELETE FROM INCUMBENTS
WHERE  SSN = 111223333
    AND  PCN = 999071
```

logical deletion



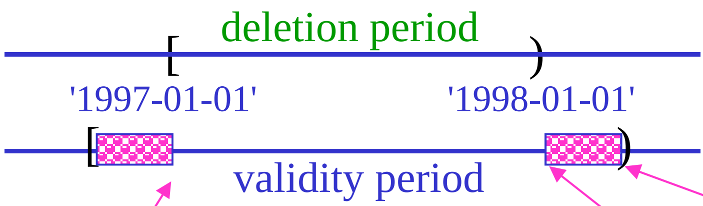
How do the physical modifications implementing the sequenced version look like?

As discussed earlier, there are again four cases to be considered, reflecting how the
period of applicability — of the deletion (here: all of 1997) and the
period of validity — of the row to be deleted
are related to each other:

1. The validity period „covers" the deletion period (during, starts, finishes, equals).
2. The validity period overlaps the deletion period (Allen overlaps).
3. The deletion period overlaps the validity period.
4. The deletion period „covers" the validity period.

In each of the cases, a different physical implementation of the logical sequenced deletion is necessary.

1. The validity period „covers“ the deletion period (during^{-1}, starts^{-1}, finishes^{-1}).

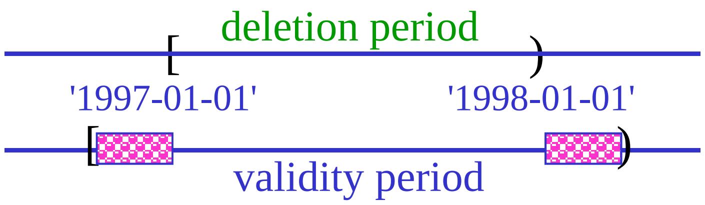



```
INSERT   INTO INCUMBENTS
         SELECT   SSN, PCN, DATE '1998-01-01', END_DATE
         FROM     INCUMBENTS
         WHERE    SSN = 111223333
            AND   PCN = 999071
            AND   START_DATE < DATE '1997-01-01'
            AND   END_DATE > DATE '1998-01-01'
```

```
UPDATE   INCUMBENTS
SET      END_DATE = DATE '1997-01-01'
WHERE    SSN = 111223333
   AND   PCN = 999071
   AND   START_DATE < DATE '1997-01-01'
   AND   END_DATE > DATE '1998-01-01'
```

(This formulation applies to the during case only, cases where period limits coincide to be considered analogously.)

Sequenced Deletions (2a)



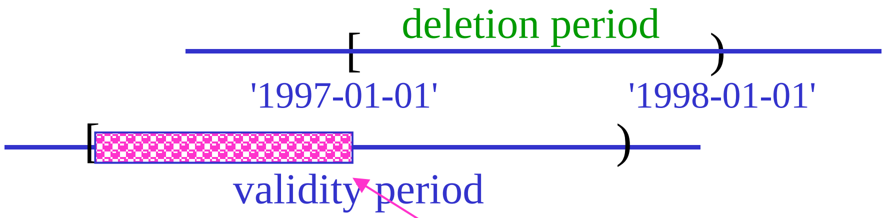
- Attention! When dealing with valid time, „the database does forget“ deleted data, as they are considered the result of erroneous information about the real happenings in the application domain.

- The part of the validity period of the „old“ row covered by the deletion period is lost!

```
UPDATE     INCUMBENTS
SET        END_DATE = DATE '1997-01-01'
WHERE      SSN = 111223333
   AND     PCN = 999071
   AND     START_DATE < DATE '1997-01-01'
   AND     END_DATE > DATE '1998-01-01'
```

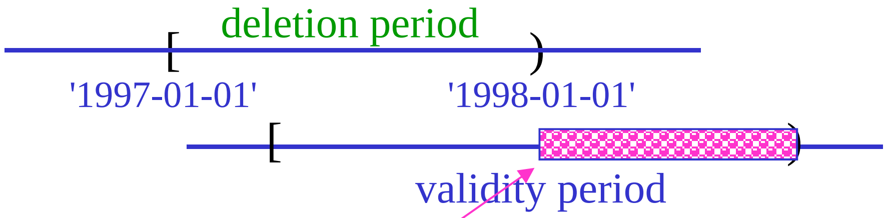
```
INSERT     INTO INCUMBENTS
           SELECT     SSN, PCN, DATE '1998-01-01', END_DATE
           FROM       INCUMBENTS
           WHERE      SSN = 111223333
              AND     PCN = 999071
              AND     START_DATE < DATE '1997-01-01'
              AND     END_DATE > DATE '1998-01-01'
```

2. The validity period overlaps the deletion period (Allen overlaps).



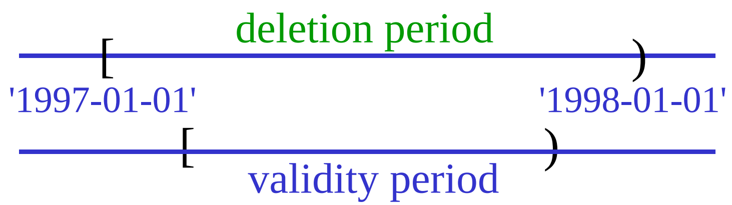

```
UPDATE INCUMBENTS
SET       END_DATE = DATE '1997-01-01'
WHERE  SSN = 111223333
    AND   PCN = 999071
    AND   START_DATE < DATE '1997-01-01'
    AND   END_DATE > DATE '1997-01-01‘
    AND   END_DATE < DATE '1998-01-01'
```

3. The deletion period overlaps the validity period.




```
UPDATE INCUMBENTS
SET     START_DATE = DATE '1998-01-01'
WHERE  SSN = 111223333
   AND  PCN = 999071
   AND  START_DATE < DATE '1998-01-01'
   AND  START_DATE >= DATE '1997-01-01'
   AND  END_DATE > DATE '1998-01-01'
```

4. The deletion period „covers“ the validity period.




```
DELETE FROM INCUMBENTS
WHERE  SSN = 111223333
   AND  PCN = 999071
   AND  START_DATE > DATE '1997-01-01'
   AND  END_DATE < DATE '1998-01-01'
```

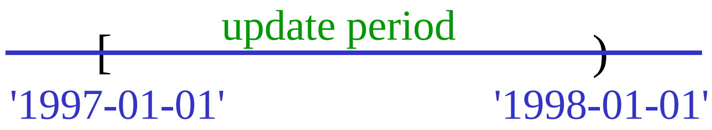
(Again, we just discuss the *during* variant here.)

Sequenced Updates (1)

Next consider applying an update (promotion of an employee) retroactively for a particular period in the past only, e.g., again for the year 1997:

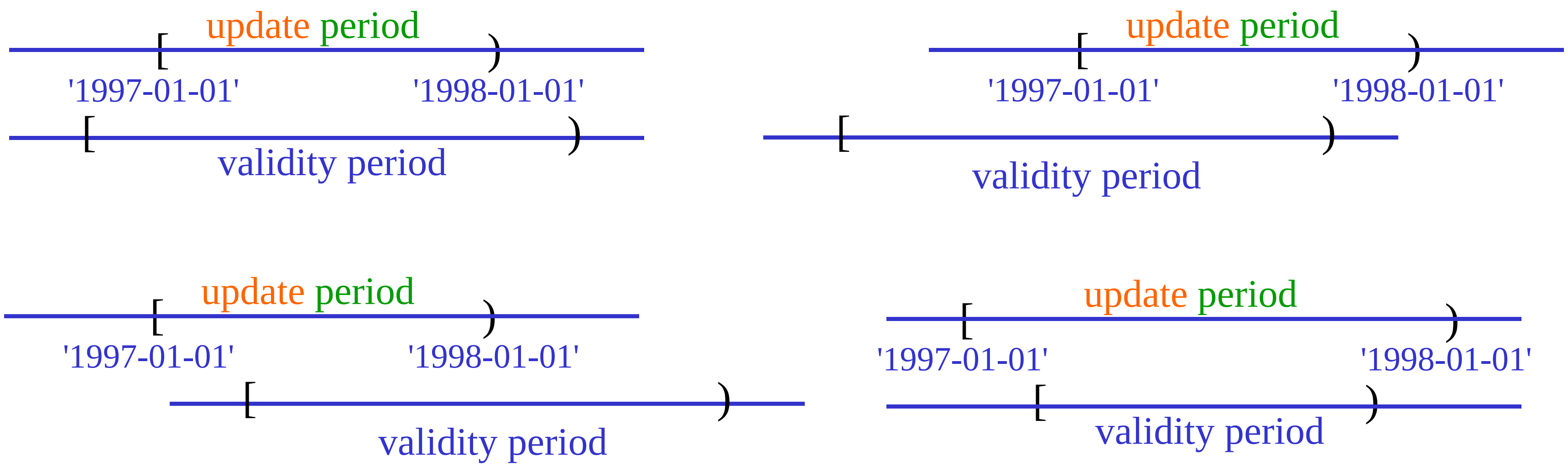
```
UPDATE INCUMBENTS
SET      PCN = 908739
WHERE    SSN = 111223333
```

logical update

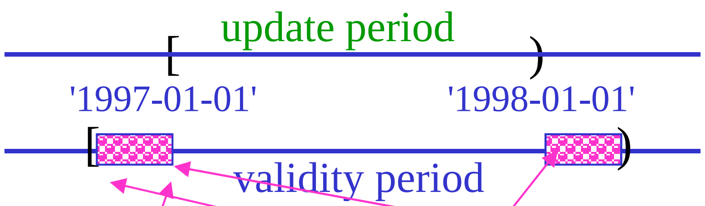


How do the physical modifications implementing the sequenced version look like?

Again, the same four cases have to be distinguished as for sequenced deletions before:



Only case 1 discussed here – similar considerations needed for cases 2 to 4:



Similar insertion needed for retaining old position for rest of validity period

```
INSERT   INTO INCUMBENTS
         SELECT  SSN, PCN, START_DATE, DATE '1997-01-01',
         FROM    INCUMBENTS
         WHERE   SSN = 111223333
           AND   PCN = 908739
           AND   START_DATE < DATE '1997-01-01'
           AND   END_DATE > DATE '1998-01-01'
```

Retains old position for first part of validity period

```
UPDATE   INCUMBENTS
SET      PCN=908739
WHERE    SSN = 111223333
  AND    START_DATE < DATE '1998-01-01'
  AND    END_DATE > DATE '1997-01-01'
```

Again, the following slides on SQL:2011 have been taken from this tutorial available online.

SQL:2011 Terminology: Reminder

Research Terminology	**SQL:2011 Terminology**
valid time	application time
transaction time	system time

timestamping	versioning

valid time table	application time period table
transaction time table	system-versioned table
bitemporal table	system-versioned application time period table

SQL:2011: Application Time Period Tables

- Application-time period tables are tables that contain a **PERIOD** clause (newly-introduced) with an user-defined period name.
- Currently restricted to temporal periods only; may be relaxed in the future.
- Application-time period tables must contain two additional columns, one to store the start time of a period associated with the row and one to store the end time of the period.
- Values of both start and end columns are set by the users.
- Additional syntax is provided for users to specify primary key/unique constraints that ensure no two rows with the same key value have overlapping periods.
- Additional syntax is provided for users to specify referential constraints that ensure the period of every child row is completely contained in the period of exactly one parent row or in the combined period of two or more consecutive parent rows.
- Queries, inserts, updates and deletes on application-time period tables behave exactly like queries, inserts, updates and deletes on regular tables.
- Additional syntax is provided on **UPDATE** and **DELETE** statements for partial period updates and deletes.

Creating an application time period table:

```
CREATE TABLE employees
(emp_name VARCHAR(50) NOT NULL PRIMARY KEY,
dept_id VARCHAR(10),
start_date DATE NOT NULL,
end_date DATE NOT NULL,
PERIOD FOR emp_period (start_date, end_date),
PRIMARY KEY (emp_name, emp_period WITHOUT OVERLAPS),
FOREIGN KEY (dept_id, PERIOD emp_period) REFERENCES
                    departments (dept_id, PERIOD dept_period));
```

The PERIOD FOR clause contains an implicit constraint (enforced by the DBMS), *CHECK start_date < end_date*. The same holds for system time.

(example from K. Kulkarni „Temporal Features in SQL Standard“)

SQL:2011: Application Time Period Tables (2)

Inserting rows into an application time period table – period values provided by users:

```
INSERT INTO employees (emp_name, dept_id, start_date, end_date)
VALUES ('John', 'J13', DATE '1995-11-15', DATE '1996-11-15'),
       ('Tracy','K25', DATE '1996-01-01', DATE '1997-11-15)
```

emp_name	dept_id	start_date	end_date
John	J13	11/15/1995	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

(example from K. Kulkarni „Temporal Features in SQL Standard“)

Updating fields in an application time period table – timestamps not affected:

emp_name	dept_id	start_date	end_date
John	J13	11/15/1995	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

```
UPDATE employees
SET dept_id = 'J15"
WHERE emp_name = 'John'
```

Timestamp unchanged!

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

(example from K. Kulkarni „Temporal Features in SQL Standard")

SQL:2011: Application Time Period Tables (4)

Updating fields in an application time period table – timestamps updated <u>too</u>:

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

```
UPDATE employees FOR PORTION OF emp_period FROM
          DATE '1996-03-01' TO DATE '1996-07-01'
SET dept_id = 'M12"
WHERE emp_name = 'John'
```

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	03/01/1996
John	M12	03/01/1996	07/01/1996
John	J15	07/01/1996	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

Automatic „row splitting" – a sequenced update!

(example from K. Kulkarni „Temporal Features in SQL Standard")

SQL:2011: Application Time Period Tables (5)

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	03/01/1996
John	M12	03/01/1996	07/01/1996
John	J15	07/01/1996	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

Deleting rows from an application time period table – a sequenced deletion „cutting" out one month from history

```
DELETE FROM employees FOR PORTION OF emp_period FROM
        DATE '1996-08-01' TO DATE '1996-09-01'
WHERE emp_name = 'John'
```

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	03/01/1996
John	M12	03/01/1996	07/01/1996
John	J15	07/01/1996	08/01/1996
John	J15	09/01/1996	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

(example from K. Kulkarni „Temporal Features in SQL Standard")

SQL:2011: Application Time Period Tables (6)

emp_name	dept_id	start_date	end_date
John	J15	11/15/1995	03/01/1996
John	M12	03/01/1996	07/01/1996
John	J15	07/01/1996	08/01/1996
John	J15	09/01/1996	11/15/1996
Tracy	K25	01/01/1996	11/15/1997

```
DELETE FROM employees
WHERE EmpName = 'John'
```

emp_name	dept_id	start_date	end_date
Tracy	K25	01/01/1996	11/15/1997

Deleting rows from an application time period table – a nonsequenced deletion eliminating all rows about John

(example from K. Kulkarni „Temporal Features in SQL Standard“)

SQL:2011: Application Time Period Tables (7)

Querying an application time period table – an application time timeslice (past) query:

employees

emp_name	dept_id	start_date	end_date
John	M24	1998-01-31	9999-12-31
John	J13	1995-11-15	1998-01-31
Tracy	K25	01/01/1996	2000-03-31

1. Which department was John in on Dec. 1, 1997?

J13

```
SELECT dept_id
FROM employees
WHERE emp_name = 'John' AND start_date <= DATE '1997-12-01' AND
end_date > DATE '1997-12-01';
```

(example from K. Kulkarni „Temporal Features in SQL Standard“)

SQL:2011: Application Time Period Tables (8)

Querying an application time period table – an application time current query:

employees

emp_name	dept_id	start_date	**end_date**
John	M24	1998-01-31	9999-12-31
John	J13	1995-11-15	1998-01-31
Tracy	K25	01/01/1996	2000-03-31

1. Which department is John in currently?

```
SELECT dept_id
FROM employees
WHERE emp_name = 'John' AND start_date <= CURRENT_DATE AND end_date >
CURRENT_DATE;
```

M24

(example from K. Kulkarni „Temporal Features in SQL Standard“)

SQL:2011: Application Time Period Tables (9)

Querying an application time period table – an application time sequenced query:

employees

emp_name	dept_id	start_date	end_date
John	M24	1998-01-31	9999-12-31
John	J13	1995-11-15	1998-01-31
Tracy	K25	01/01/1996	2000-03-31

1. How many departments has John worked in since Jan. 1, 1996?

```
SELECT count(distinct dept_id)
FROM employees WHERE emp_name = 'John' AND start_date <= CURRENT_DATE
AND end_date > DATE '1996-01-01';
```

2

(example from K. Kulkarni „Temporal Features in SQL Standard“)

SQL:2011: Application Time Period vs. System-Versioned Tables

```
CREATE TABLE employees
(emp_name VARCHAR(50) NOT NULL,
dept_id VARCHAR(10),
system_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
system_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME (system_start, system_end),
PRIMARY KEY (emp_name),
FOREIGN KEY (dept_id) REFERENCES  departments (dept_id);
)  WITH SYSTEM VERSIONING;
```

Declaring a system-versioned table

Declaring an application time period table

```
CREATE TABLE employees
(emp_name VARCHAR(50) NOT NULL PRIMARY KEY,
dept_id VARCHAR(10),
start_date DATE NOT NULL,
end_date DATE NOT NULL,
PERIOD FOR emp_period (start_date, end_date),
PRIMARY KEY (emp_name, emp_period WITHOUT OVERLAPS),
FOREIGN KEY (dept_id, PERIOD emp_period) REFERENCES
                         departments (dept_id, PERIOD dept_period));
```

SQL:2011: Coming Close to Allen's Operators

- In order to „simplify" the formulation of conditions involving time-valued attributes in SQL, new operators and keywords have been introduced in SQL:2011 …
  - … coming close to the Allen operators for comparing periods, without following Allen's terminology
  - … extending the already existing SQL operator OVERLAPS
  - … introducing a new style for expressing period expressions (without introducing a new datatype PERIOD).

- PRECEDES corresponds to Allen's *before*
  IMMEDIATELY PRECEDES corresponds to Allen's *meets*
  EQUALS corresponds to Allen's *equals*
  CONTAINS corresponds to Allen's *during*
  (special form for periods with just one instant: no period notation necessary)
  IMMEDIATELY SUCCEEDS corresponds to Allen's $meets^{-1}$
  SUCCEEDS corresponds to Allen's $before^{-1}$

- OVERLAPS retains its previously established meaning.

- Bracketed operands of these operators are now pre-fixed by the keyword PERIOD, e.g., PERIOD (CURRENT_DATE, CURRENT_DATE + 3 DAY)

SQL:2011: Application Time Period vs. System-Versioned Tables

```
CREATE TABLE employees
(emp_name VARCHAR(50) NOT NULL,
dept_id VARCHAR(10),
system_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
system_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME (system_start, system_end),
PRIMARY KEY (emp_name),
FOREIGN KEY (dept_id) REFERENCES departments (dept_id);
) WITH SYSTEM VERSIONING;
```

Declaring a system-versioned table

Declaring an application time period table

```
CREATE TABLE employees
(emp_name VARCHAR(50) NOT NULL PRIMARY KEY,
dept_id VARCHAR(10),
start_date DATE NOT NULL,
end_date DATE NOT NULL,
PERIOD FOR emp_period (start_date, end_date),
PRIMARY KEY (emp_name, emp_period WITHOUT OVERLAPS),
FOREIGN KEY (dept_id, PERIOD emp_period) REFERENCES
                    departments (dept_id, PERIOD dept_period));
```

Modifications for Application Time: Summary

Application Time timestamps to be explicitly given at row insertion:

```
INSERT INTO employees (emp_name, dept_id, start_date, end_date)
VALUES    ('John', 'J13', DATE '1996-11-15', DATE '1997-11-15''),
```

New syntax just for application time UPDATE / DELETE:
FOR PORTION OF ….

```
UPDATE employees
        FOR PORTION OF emp_period
        FROM DATE '1996-03-01' TO DATE '1996-07-01'
SET dept_id = 'M12''
WHERE emp_name = 'John'
```

```
DELETE FROM employees
        FOR PORTION OF emp_period
        FROM DATE '1996-08-01' TO DATE '1996-09-01'
WHERE emp_name = 'John'
```

Modifications for System Time: Summary

```
INSERT INTO  emp (emp_name, dept_id)
VALUES (‘John’, ’J13’)
```

System time values initiated by the DBMS.

No new syntax for any system time modification, but automated modification of system time values

```
UPDATE  emp
SET   dept_id = ‘M24’’
WHERE  emp_name = ‘John’
```

```
DELETE FROM  emp
WHERE  emp_name = ‘Tracy’
```

Physical modifications of system time attributes done automatically by DBMS! Syntax of commands represents logical modifications only.

Queries for Application Time: Summary

No new syntax for any application time query!

```
SELECT dept_id
FROM  employees
WHERE  emp_name = ‘John’
        AND  start_date  <= DATE ‘1997-12-01’
        AND  end_date    >   DATE ‘1997-12-01’ ;
```

Temporal condition to be explicitly included into WHERE-part.

But: New period comparison operators can be used (sometimes simplifying effort), e.g., using overloaded CONTAINS for time-slice queries and period name for application time PERIOD declarations.

```
… AND  emp_period   CONTAINS   DATE ‘1997-12-01’ ;
```

Queries for System Time: Summary

time-slice (past) query:

New syntax for all system time queries:
FOR SYSTEM_TIME ….

```
SELECT Dept
FROM employees
        FOR SYSTEM_TIME  AS OF  DATE '1997-12-01'
WHERE emp_name = 'John'
```

AS OF

sequenced query:

```
SELECT count(distinct dept_id)
FROM employees
        FOR SYSTEM_TIME   FROM   DATE '1996-01-01'  TO   CURRENT_DATE
WHERE emp_name = 'John'
```

FROM … TO ….

SQL:2011: Queries and Modifications in Comparison

	Modifications	**Queries**
System time	No new syntax (but automated management of system time period values)	New syntax (AS OF, FROM .. TO ..)
Application time	New syntax just for UPDATE/DELETE) (FOR PORTION OF ..)	No new syntax (but new period comparison operators can be used)