Temporal Data and The Relational Model

Hugh Darwen
hugh@dcs.warwick.ac.uk
www.dcs.warwick.ac.uk/~hugh

Based on the book of the same title,
by C.J. Date, Hugh Darwen, and Nikos A. Lorentzos
summarised in C.J. Date: Introduction to Database

for
Warwick University
CS319

The Book’s Aims

- Describe a foundation for inclusion of support for temporal data in a truly relational database management system (TRDBMS).
- Focusing on problems related to data representing beliefs that hold throughout given intervals (usually, of time).
- Propose additional operators on relations and relation variables (“relvars”) having interval-valued attributes.
- Propose additional constraints on relation variables having interval-valued attributes.
- All of the above to be definable in terms of existing operators and constructs.
- And explore some interesting side issues.

Contents (Parts I and II)

Part I: Preliminaries
Chapter 1: A Review of Relational Concepts
Chapter 2: An Overview of Tutorial D

Part II: Laying the Foundations
Chapter 3: Time and the Database
Chapter 4: What Is the Problem?
Chapter 5: Intervals
Chapter 6: Operators on Intervals
Chapter 7: The COLLAPSE and EXPAND Operators
Chapter 8: The PACK and UNPACK Operators
Chapter 9: Generalising the Relational Operators

Contents (Part III)

Part III: Building on the Foundations
Chapter 10: Database Design
Chapter 11: Integrity Constraints I: Candidate Keys and Related Constraints
Chapter 12: Integrity Constraints II: General Constraints
Chapter 13: Database Queries
Chapter 14: Database Updates
Chapter 15: Stated Times and Logged Times
Chapter 16: Point and Interval Types Revisited

Part I: Preliminaries

Chapter 1: A Review of Relational Concepts
Introduction: The running example (based on Date’s familiar “suppliers and parts” database); Types; Relation values; Relation variables; Integrity constraints; Relational operators; The relational model; Exercises (as for every chapter).

Chapter 2: An Overview of Tutorial D
A relational database language devised for tutorial purposes by Date and Darwen in “Databases, Types, and The Relational Model: The Third Manifesto” (3rd edition, Addison-Wesley, 2005). Also used in 8th edition of Date’s “Introduction to Database Systems”.

Introduction; Scalar type definitions; Relational definitions; Relational expressions; Relational assignments; Constraint definitions; Exercises.
Chapter 3: Time and the Database

Introduction

Timestamped propositions
E.g. "Supplier S1 was under contract throughout the period from 1/9/1999 (and not immediately before that date) until 31/5/2002 (and not immediately after that date)."

"Valid time" vs. "transaction time"
Some fundamental questions:
Introduction of quantisation and its consequences.

Example: Current State Only

"Suppliers and Shipments"

Consider queries:
Which suppliers can supply something? Which suppliers cannot supply anything?

"Semitemporalising"

Consider queries:
Since when has supplier S# been able to supply anything? (Not too difficult)
Since when has supplier S# been unable to supply anything? (Impossible)

"Fully temporalising" (try 1)

Consider queries:
During which times was supplier S# able to supply anything? (Very difficult)
During which times was supplier S# unable to supply anything? (Very difficult)

Interpretation? Are those FROM and TO dates included? E.g. Was supplier S1 under contract on day 10?

CHAPTER 4: What is The Problem?

"Required Constraints"

Same supplier can't be under contract during distinct but overlapping or abutting intervals.

These are very difficult!
CHAPTER 5: Intervals

"Fully temporalising" (try 2)

<table>
<thead>
<tr>
<th>S_DURING</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>d04</td>
<td>d11</td>
</tr>
<tr>
<td>S2</td>
<td>d02</td>
<td>d05</td>
</tr>
<tr>
<td>S2</td>
<td>d07</td>
<td>d11</td>
</tr>
<tr>
<td>S3</td>
<td>d03</td>
<td>d11</td>
</tr>
<tr>
<td>S4</td>
<td>d04</td>
<td>d11</td>
</tr>
<tr>
<td>S5</td>
<td>d03</td>
<td>d11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SP_DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>S5</td>
</tr>
</tbody>
</table>

Consider queries: During which times was supplier S# able to supply anything? (Very difficult)
During which times was supplier S# unable to supply anything? (Very difficult)

Interpretation? FROM dates are included, TO dates are not included: the "closed-open" convention.

"Fully temporalising" in SQL:2011

<table>
<thead>
<tr>
<th>S_DURING</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>d04</td>
<td>d11</td>
</tr>
<tr>
<td>S2</td>
<td>d02</td>
<td>d05</td>
</tr>
<tr>
<td>S2</td>
<td>d07</td>
<td>d11</td>
</tr>
<tr>
<td>S3</td>
<td>d03</td>
<td>d11</td>
</tr>
<tr>
<td>S4</td>
<td>d04</td>
<td>d11</td>
</tr>
<tr>
<td>S5</td>
<td>d03</td>
<td>d11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SP_DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
</tr>
<tr>
<td>S5</td>
</tr>
</tbody>
</table>

Consider queries: During which times was supplier S# able to supply anything? (Very difficult)
During which times was supplier S# unable to supply anything? (Very difficult)

Interpretation? FROM dates are included, TO dates are not included: the "closed-open" convention.

CHAPTER 6: Operators on Intervals

Interval Selectors

In Tutorial D, we make the type name part of the operator name. E.g.:

```
INTERVAL_INTEGER ( [1:10] )
```

Note special syntax for denoting bounds. Square bracket denotes a closed bound, round one an open bound. Thus:

```
INTERVAL_INTEGER ( [1:10] ) == INTERVAL_INTEGER ( (0:10] )
INTERVAL_INTEGER ( [1:11) ) == INTERVAL_INTEGER ( 1:11 ]
```

So the problem of interpretation does not arise.

Interval Selectors in SQL:2011

SQL:2011 does not support intervals.

Interval types are not to be confused with INTERVAL types, that have been in standard SQL since 1992.

An INTERVAL value in SQL denotes a duration, not an interval! (So we will say no more about SQL INTERVAL types.)
Monadic Operators on Intervals

For a given interval, \( i \):
- \( \text{PRE}(i) \) gives open begin bound
- \( \text{BEGIN}(i) \) gives closed begin bound
- \( \text{END}(i) \) gives closed end bound
- \( \text{POST}(i) \) gives open end bound
- \( \text{COUNT}(i) \) gives length (number of points)

SQL:2011 Counterparts of Monadic Operators on Intervals

For a given from-to pair, \(<f, t>\), representing interval \( i \), where \( f \) and \( t \) are expressions of type DATE (for example):

<table>
<thead>
<tr>
<th>Tutorial D</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{PRE}(i) )</td>
<td>( f - 1 ) DAY</td>
</tr>
<tr>
<td>( \text{BEGIN}(i) )</td>
<td>( f )</td>
</tr>
<tr>
<td>( \text{END}(i) )</td>
<td>( t - 1 ) DAY</td>
</tr>
<tr>
<td>( \text{POST}(i) )</td>
<td>( t )</td>
</tr>
<tr>
<td>( \text{COUNT}(i) )</td>
<td>( t - f )</td>
</tr>
</tbody>
</table>

Comparisons of Two Intervals

For given intervals, \( i_1 \) and \( i_2 \):
- \( i_1 = i_2 \)
- \( i_1 \text{ MEETS } i_2 \)
- \( i_1 \text{ OVERLAPS } i_2 \)
- \( i_1 \text{ SUCCEEDS } i_2 \)
- \( i_1 \text{ PRECEDES } i_2 \)
- \( i_1 \text{ BEGINS } i_2 \)
- \( i_1 \text{ ENDS } i_2 \)
- \( i_1 \text{ MERGES } i_2 \)

Allen's operators
(James F. Allen, 1983)
- Allen uses \textit{DURING} for \( \subseteq \)
- Allen uses \textit{STARTS} and \textit{ENDS}

Added by Date, Darwen, Lorentzos
- MERGES = MEETS OR OVERLAPS

Some Pictorial Definitions

For given intervals, \( i_1 \) and \( i_2 \):
- \( i_1 = i_2 \)
- \( i_1 \text{ MEETS } i_2 \)
- \( i_1 \text{ OVERLAPS } i_2 \)
- \( i_1 \text{ SUCCEEDS } i_2 \)
- \( i_1 \text{ PRECEDES } i_2 \)
- \( i_1 \text{ BEGINS } i_2 \)
- \( i_1 \text{ ENDS } i_2 \)
- \( i_1 \text{ MERGES } i_2 \)

Interval Comparison in SQL:2011

For given from-to pairs, \(<f_1, t_1>\) and \(<f_2, t_2>\):

<table>
<thead>
<tr>
<th>Tutorial D</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 = t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ EQUALS } &lt;f_2, t_2&gt;)</td>
</tr>
<tr>
<td>( f_1 \text{ MEETS } t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ IMMEDIATELY PRECEDES } &lt;f_2, t_2&gt; ) OR (&lt;f_2, t_2&gt; \text{ IMMEDIATELY PRECEDES } &lt;f_1, t_1&gt;)</td>
</tr>
<tr>
<td>( f_1 \text{ OVERLAPS } t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ OVERLAPS } &lt;f_2, t_2&gt;)</td>
</tr>
<tr>
<td>( f_1 \text{ SUCCEEDS } t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ SUCCEEDS } &lt;f_2, t_2&gt;)</td>
</tr>
<tr>
<td>( f_1 \text{ PRECEDES } t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ PRECEDES } &lt;f_2, t_2&gt;)</td>
</tr>
<tr>
<td>( f_1 \text{ BEGINS } t_2 )</td>
<td>( f_1 = t_2)</td>
</tr>
<tr>
<td>( f_1 \text{ ENDS } t_2 )</td>
<td>( t_1 = t_2)</td>
</tr>
<tr>
<td>( f_1 \subseteq t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ CONTAINS } &lt;f_2, t_2&gt;)</td>
</tr>
<tr>
<td>( f_1 \supseteq t_2 )</td>
<td>(&lt;f_2, t_2&gt; \text{ CONTAINS } &lt;f_1, t_1&gt;) AND NOT((&lt;f_1, t_1&gt; \text{ EQUALS } &lt;f_2, t_2&gt;))</td>
</tr>
<tr>
<td>( f_1 \supseteq t_2 )</td>
<td>(&lt;f_1, t_1&gt; \text{ CONTAINS } &lt;f_2, t_2&gt;) AND NOT((&lt;f_1, t_1&gt; \text{ EQUALS } &lt;f_2, t_2&gt;))</td>
</tr>
<tr>
<td>( f_1 \text{ MERGES } t_2 )</td>
<td>Sorry, not enough room on slide! (Exercise for reader)</td>
</tr>
</tbody>
</table>

How to Express \(<f, t>\) in SQL:2011

In general:

\[
\text{PERIOD (} f, t \text{)}
\]

where \( f \) and \( t \) are DATE, TIME or TIMESTAMP expressions of the same type (i.e., same precision and scale)

N.B. PERIOD is not an operator! It's just a "noise" word.

Special case:

\[
\text{pn}
\]

where \( pn \) is a defined period name (see later)
Comment on SQL:2011 IMMEDIATELY operators

Consider:

\[
\text{PERIOD (} f_1, t_1 \text{) IMMEDIATELY PRECEDES} \\text{PERIOD (} f_2, t_2 \text{)}
\]

Is this a handy addition to the language in SQL:2011?

Well, we have always been able to write \( t_1 = f_2 \) !

Allen’s MEETS seems rather more useful.

More Dyadic Operators

Membership test:

\[
p \in i \quad \text{or} \quad p \in i (\text{where } p \text{ is a point})
\]

\[
i \text{ CONTAINS } p \text{ in SQL:2011}
\]

Dyadic operators that return intervals:

\[
\begin{align*}
&i_1 \text{ UNION } i_2 \\
&i_1 \text{ INTERSECT } i_2 \\
&i_1 \text{ MINUS } i_2
\end{align*}
\]

* empty intervals, such as INTERVAL_INTEGER ([1:1]), are not supported at all!

No direct counterparts of UNION, INTERSECT, MINUS in SQL:2011.

Sets of Intervals

Let \( S_1 \) and \( S_2 \) be sets of intervals—e.g., \([1:2], [4:7], [6:9]\)\]

We define an equivalence relationship:

\[
S_1 = S_2 \text{ iff every point in an interval in } S_1 \text{ is a point in some interval in } S_2, \text{ and vice versa.}
\]

Under this equivalence relationship we then define two canonical forms: **collapsed form** and **expanded form**.

In each of these forms, no point appears more than once.

Collapsed Form

No two elements, \( i_1 \) and \( i_2 \) (\( i_1 \neq i_2 \)) are such that

\[
i_1 \text{ MERGES } i_2.
\]

So the collapsed form of \([1:2], [4:7], [6:9]\) is \([1:2], [4:9]\).

Expanded Form

Every element is a **unit interval**

(i.e., consists of a single point)

So the expanded form of \([1:2], [4:7], [6:9]\)

is \([1:1], [2:2], [4:4], [5:5], [6:6], [7:7], [8:8], [9:9]\).
COLLAPSE and EXPAND

Let $S$ be a set of intervals.

Then:
COLLAPSE($S$) denotes the collapsed form of $S$.
EXPAND($S$) denotes the expanded form of $S$.

These operators are handy for definitional purposes (as we shall see) but are not required to exist in the database language.

CHAPTER 8: The PACK and UNPACK Operators

Packed Form and Unpacked Form

Canonical forms for relations with one or more interval-valued attributes.

Based on collapsed and expanded forms.

Both forms avoid redundancy ("saying the same thing" more than once).

Packed Form

Unpacked Form

Properties of PACK and UNPACK

Packing and unpacking on no attributes:
• Important degenerate cases
• Each yields its input relation

Unpacking on several attributes:
• UNPACK $r$ ON $(a_1, a_2)$ =
  UNPACK (UNPACK $r$ ON $a_1$) ON $a_2$ =
  UNPACK (UNPACK $r$ ON $a_2$) ON $a_1$

Packing on several attributes:
• PACK $r$ ON $(a_1, a_2)$ =
  PACK (UNPACK $r$ ON $(a_1, a_2)$) ON $a_1$ ON $a_2$
  not: PACK (PACK(UNPACK $r$ ON $(a_1, a_2)$) ON $a_1$) ON $a_2$
  and not: PACK (PACK $r$ ON $a_1$) ON $a_2$

• Although redundancy is eliminated, result can be of greater cardinality than $r$. 
Packed and Unpacked Form in SQL:2011

- SQL:2011 does not support a PACK operator
- SQL:2011 does not support an UNPACK operator

Even though both were once (in the 1990s) included in Part 7, SQL/Temporal, a working draft that was never published and eventually abandoned.

CHAPTER 9:
Generalizing the Relational Operators

Tutorial D’s Relational Operators

UNION
MATCHING
NOT MATCHING
restriction (WHERE)
projection ({{...}})
JOIN
EXTEND
SUMMARIZE
etc.

New syntax for invoking each operator:

USING (ACL) `rel op inv`

where ACL is an attribute-name commalist and `rel op inv` an invocation of a relational operator.

Common semantics:
1. Unpack the operand(s) on ACL
2. Evaluate rel op inv on unpacked forms.
3. Pack result of 2. on ACL

USING Example 1

USING (DURING) `SP_DURING (S#, DURING)`

gives (S#, DURING) pairs such that supplier S# was able to supply some part throughout the interval DURING.

We call this "U_project".
U_project is an example of what we call a "U_ operator".
Other examples are U_JOIN, U_UNION, U_restrict, etc.

Example 2: U_NOT MATCHING

USING (DURING) `S_DURING NOT MATCHING SP_DURING`

gives (S#, DURING) pairs such that supplier S# was under contract but unable to supply any part throughout the interval DURING.

Note: We have now solved the two query problems mentioned in Chapter 4, "What’s the Problem?"

Example 3: U_SUMMARIZE

USING (DURING)

SUMMARIZE SP_DURING 
PER (S_DURING {S#, DURING}):
{NO_OF_PARTS := COUNT ()} 

gives (S#, NO_OF_PARTS, DURING) triples such that supplier S# was able to supply NO_OF_PARTS parts throughout the interval DURING.

Temporal counterpart of:

SUMMARIZE SP PER (S {S#}):
{NO_OF_PARTS := COUNT ( )}
U_SUMMARIZE is Interesting (1)

USING ( DURING )
SUMMARIZE SP_DURING
PER ( S_DURING { DURING } ):
{ NO_OF_PARTS := COUNT( ) }★

• note lack of S# from PER relation
• gives (NO_OF_PARTS, DURING) pairs such that
NO_OF_PARTS parts were available from some supplier
throughout the interval DURING.

U_SUMMARIZE is Interesting (2)

USING ( DURING )
SUMMARIZE SP_DURING
PER ( S_DURING { S# } ):
{ NO_OF_CASES := COUNT ( ) }★

• note lack of DURING from PER relation
• gives (S#, NO_OF_CASES) pairs such that there are
NO_OF_CASES distinct cases of S# being able to supply
some part on some date.

USING in SQL:2011

• SQL:2011 does not support USING

CHAPTER 10: Database Design

Current Relvars Only

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>STATUS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>10</td>
<td>Paris</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>30</td>
<td>Paris</td>
</tr>
<tr>
<td>S4</td>
<td>Clark</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>S5</td>
<td>Adams</td>
<td>30</td>
<td>Athens</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SP</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P4</td>
</tr>
<tr>
<td>S2</td>
<td>P3</td>
</tr>
<tr>
<td>S3</td>
<td>P2</td>
</tr>
<tr>
<td>S4</td>
<td>P4</td>
</tr>
<tr>
<td>S5</td>
<td>P5</td>
</tr>
</tbody>
</table>

Note: keys indicated by underlining attribute names

Contents

Chapter 10: Database Design
• Introduction
• Current relvars only
• Historical relvars only
• Sixth normal form (6NF)
• “The moving point now”
• Both current and historical relvars
• Concluding remarks
• Exercises

At last, we focus on specifically temporal issues!
Problem: SINCE gives date of last update for that supplier.
So we cannot tell:
- since when a given supplier’s STATUS has held,
- since when a given supplier’s CITY has held, or
- since when a given supplier’s NAME has held, or even
- since when a given supplier has been under contract.

Predicate:
Supplier S# has been under contract since S#_SINCE,
has been named NAME since NAME_SINCE,
has had status STATUS since STATUS_SINCE and
has been located in city CITY since CITY_SINCE.
But we clearly cannot develop a fully temporalized
counterpart on similar lines!

“As Circumlocution” and 6NF
Note S1 named Smith throughout [d01:d09], split across tuples. We call this possibly undesirable phenomenon circumlocution. Decompose to 6NF, using U_projection:

And so on. We call this process vertical decomposition.

“Circumlocution” and 6NF

Fully Temporalizing SSSC

VAR S_DURING
BASE RELATION
( S# S#, DURING INTERVAL_DATE )
PERIOD FOR DURING (S#_FROM, S#_TO)
KEY ( S#, DURING ) ;

VAR S_NAME_DURING
BASE RELATION
( S# S#, SNAME CHAR, DURING INTERVAL_DATE )
PERIOD FOR DURING (SNAME_FROM, SNAME_TO)
KEY ( S#, DURING ) ;

And so on. We call this process vertical decomposition.

Sixth Normal Form (6NF)
Recall: A relvar R is in 5NF iff every nontrivial join dependency that is satisfied by R is implied by a candidate key of R.

A relvar R is in 6NF iff R satisfies no nontrivial join dependencies at all (in which case R is sometimes said to be irreducible).

SSSC and SSSC_SINCE are in 5NF but not 6NF (which is not needed).
S_DURING, S_NAME_DURING and so on are in 6NF, thus allowing each of the supplier properties NAME, CITY and STATUS, which vary independently of each other over time, to have its own recorded history (by supplier).

Fully Temporalizing SSSC in SQL:2011
CREATE TABLE S_DURING
( S# S#, S#_FROM DATE, S#_TO DATE, PERIOD FOR DURING (S#_FROM, S#_TO), PRIMARY KEY ( S#, S#_FROM, S#_TO ) ;

CREATE TABLE S_NAME_DURING
( S# S#, SNAME VARCHAR(50), SNAME_FROM DATE, SNAME_TO DATE, PERIOD FOR DURING (SNAME_FROM, SNAME_TO), PRIMARY KEY ( S#, SNAME_FROM, SNAME_TO ) ;

And so on. No more than one application time period name per base table.
Using SQL:2011 Period Names in Queries

E.g., to find pairs of suppliers who were under contract at the same time:

```sql
SELECT S1.S# AS S#1, S1.S#_FROM AS F1, S1.S#_TO AS T1
FROM S_DURING S1, S_DURING S2
WHERE S1.DURING OVERLAPS S2.DURING
```

Note:
- can’t use period names in SELECT clause
- period names not defined for result, so are lost when any subquery referencing S_DURING is used in a FROM clause or a view definition

“The Moving Point NOW”

We reject any notion of a special marker, NOW, as an interval bound. (It is a variable, not a value. Its use would be as much a departure from the Relational Model as NULL is!)

(We reject the use of NULL too, obviously.)

If current state is to be recorded, along with history, in S_DURING, S_NAME_DURING, S_STATUS_DURING and S_CITY_DURING, then we have a choice of evils:
- guess when, in the future, current state will change
- assume current state will hold until the end of time

Better instead to use horizontal decomposition

Horizontal Decomposition

A very loose term! Components do not have exactly the same structure:

1. The current state component (S_SINCE)
2. The past history component, with DURING in place of S_SINCE’S SINCE.

The past history component is then vertically decomposed as already shown, giving S_DURING, S_NAME_DURING, S_STATUS_DURING, and S_CITY_DURING.

Having accepted the occasional (perhaps frequent) inevitability of vertical and horizontal decomposition, we need to consider the consequences for constraints ...

“The Moving Point NOW” in SQL:2011

NULL is not used. Hooray! for that.

SQL uses “the end of time”. So what’s that in SQL?

23:59:59.999999 on December 31, 9999

Candidate Keys and Related Constraints

Example database:

```
S_SINCE ( S#, S#_SINCE, STATUS, STATUS_SINCE )
SP_SINCE ( S#, P#, SINCE )
S_DURING ( S#, DURING )
S_STATUS_DURING ( S#, STATUS, DURING )
SP_DURING ( S#, P#, DURING )
```

We first examine three distinct problems:
- The redundancy problem
- The circumlocution problem
- The contradiction problem

A fourth problem, concerning “density”, will come later.
The Redundancy Problem

Consider:

\[ S\_STATUS\_DURING \{ S\#, STATUS, DURING \} \]

The declared key, \( \{ S\#, DURING \} \) doesn't prevent this:

<table>
<thead>
<tr>
<th>S#</th>
<th>STATUS</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4</td>
<td>25</td>
<td>[d05 : d06]</td>
</tr>
<tr>
<td>S4</td>
<td>25</td>
<td>[d06 : d07]</td>
</tr>
</tbody>
</table>

S4 shown twice as having status 25 on day 6.
Avoided in the packed form of \( S\_STATUS\_DURING \).

The Circumlocution Problem

Still considering:

\[ S\_STATUS\_DURING \{ S\#, STATUS, DURING \} \]

The declared key, \( \{ S\#, DURING \} \) doesn't prevent this:

<table>
<thead>
<tr>
<th>S#</th>
<th>STATUS</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4</td>
<td>25</td>
<td>[d05 : d06]</td>
</tr>
<tr>
<td>S4</td>
<td>25</td>
<td>[d06 : d07]</td>
</tr>
</tbody>
</table>

Longwinded way of saying that S4 has status 25 from day 5 to day 7.
Also avoided in the packed form of \( S\_STATUS\_DURING \).

Solving The Redundancy and Circumlocution Problems

VAR \( S\_STATUS\_DURING \) RELATION
\( \{ S\#, STATUS, DURING\} \)
KEY \( \{ S\#, DURING \} \)
PACKED ON \( (DURING) \);

PACKED ON \( (DURING) \) causes an update to be rejected if acceptance would result in \( S\_STATUS\_DURING \neq \) PACK \( S\_STATUS\_DURING \) ON \( (DURING) \)

This kills two birds with one stone. We see no compelling reason for distinct shorthands to separate the two required constraints.

Solving The Contradiction Problem

\[ S\_STATUS\_DURING \{ S\#, STATUS, DURING \} \]

The declared key, \( \{ S\#, DURING \} \) and PACKED ON \( (DURING) \) don't prevent this:

<table>
<thead>
<tr>
<th>S#</th>
<th>STATUS</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4</td>
<td>25</td>
<td>[d04 : d06]</td>
</tr>
<tr>
<td>S4</td>
<td>10</td>
<td>[d05 : d07]</td>
</tr>
</tbody>
</table>

S4 has two statuses on days 5 and 6.
Easily avoidable in the unpacked form of \( S\_STATUS\_DURING \)!

Solving The Contradiction Problem in SQL:2011

CREATE TABLE \( S\_STATUS\_DURING \)
\( \{ S\#, STATUS\_CHAR, DURING INTERVAL\_DATE \} \)
KEY \( \{ S\#, DURING \} \)
PACKED ON \( (DURING) \)
WHEN UNPACKED ON \( (DURING) \)
THEN KEY \( \{ S\#, DURING \} \);

WHEN UNPACKED ON \( (DURING) \) THEN KEY \( \{ S\#, DURING \} \)
causes an update to be rejected if acceptance would result in failure to satisfy a uniqueness constraint on \( \{ S\#, DURING \} \) in the result of \( UNPACK S\_STATUS\_DURING \) ON \( (DURING) \).
Solving The Circumlocution Problem in SQL:2011

SQL:2011 offers no solution to the circumlocution problem

Neither WHEN / THEN nor PACKED ON

Example (measures of inflation):

<table>
<thead>
<tr>
<th>INFLATION</th>
<th>DURING</th>
<th>PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[m01:m03]</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>[m04:m06]</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>[m07:m09]</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>[m07:m07]</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>..........</td>
<td>..</td>
<td></td>
</tr>
<tr>
<td>[m01:m12]</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

But the predicate for this is not:
"Inflation was at PERCENTAGE throughout the interval DURING"
but rather, perhaps:
"Inflation was measured to be PERCENTAGE over the interval DURING"

WHEN / THEN and PACKED ON both required

```
VAR S_STATUS_DURING RELATION
{ S#, S#, STATUS CHAR, DURING INTERVAL_DATE }
USING ( DURING ) \ KEY { S#, DURING } \ ;

USING ( ACL ) \ KEY { K } \ ; where K includes ACL, is
shorthand for:
WHEN UNPACKED ON ( ACL )
THEN KEY ( K )
PACKED ON ( ACL )
KEY ( K )

(KEY ( K ) is implied by WHEN THEN + PACKED ON anyway)
```

We call this constraint a "U_key" constraint.

General Constraints

Example database is still:
```
S_SINCE { S#, S#_SINCE, STATUS, STATUS_SINCE }
SP_SINCE { S#, P#, SINCE }
S_DURING { S#, DURING }
S_STATUS_DURING { S#, STATUS, DURING }
SP_DURING { S#, P#, DURING }
```

with added U_keys. But more constraints are needed.

We examine nine distinct requirements, in three groups of three. In each group, one requirement relates to redundancy (and sometimes also to contradiction), one to circumlocution and one to denseness.
### Requirement Group 1

<table>
<thead>
<tr>
<th>Requirement R1:</th>
<th>Requirement R2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being under contract on day &lt;sub&gt;d&lt;/sub&gt;, then it must contain exactly one tuple that shows that fact. Note: avoiding redundancy</td>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being under contract on days &lt;sub&gt;d&lt;/sub&gt; and &lt;sub&gt;d&lt;/sub&gt;+1, then it must contain exactly one tuple that shows that fact. Note: avoiding redundancy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement R3:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being under contract on day &lt;sub&gt;d&lt;/sub&gt;, then it must also show supplier S&lt;sub&gt;x&lt;/sub&gt; as having some status on day &lt;sub&gt;d&lt;/sub&gt;. Note: to do with denseness</td>
<td></td>
</tr>
</tbody>
</table>

### Requirement Group 2

<table>
<thead>
<tr>
<th>Requirement R4:</th>
<th>Requirement R5:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as having some status on day &lt;sub&gt;d&lt;/sub&gt;, then it must contain exactly one tuple that shows that fact. Note: avoiding redundancy and contradiction</td>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as having status s on days &lt;sub&gt;d&lt;/sub&gt; and &lt;sub&gt;d&lt;/sub&gt;+1, then it must contain exactly one tuple that shows that fact. Note: avoiding circumlocution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement R6:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as having some status on day &lt;sub&gt;d&lt;/sub&gt;, then it must also show supplier S&lt;sub&gt;x&lt;/sub&gt; as being under contract on day &lt;sub&gt;d&lt;/sub&gt;. Note: to do with denseness</td>
<td></td>
</tr>
</tbody>
</table>

### Requirement Group 3

<table>
<thead>
<tr>
<th>Requirement R7:</th>
<th>Requirement R8:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being able to supply part P&lt;sub&gt;y&lt;/sub&gt; on day &lt;sub&gt;d&lt;/sub&gt;, then it must contain exactly one tuple that shows that fact. Note: avoiding redundancy</td>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being able to supply part P&lt;sub&gt;y&lt;/sub&gt; on days &lt;sub&gt;d&lt;/sub&gt; and &lt;sub&gt;d&lt;/sub&gt;+1, then it must contain exactly one tuple that shows that fact. Note: avoiding circumlocution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement R9:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>If the database shows supplier S&lt;sub&gt;x&lt;/sub&gt; as being able to supply some part on day &lt;sub&gt;d&lt;/sub&gt;, then it must also show supplier S&lt;sub&gt;x&lt;/sub&gt; as being under contract on day &lt;sub&gt;d&lt;/sub&gt;. Note: to do with denseness</td>
<td></td>
</tr>
</tbody>
</table>

### Meeting the Nine Requirements (a): current relvars only

<table>
<thead>
<tr>
<th>S&lt;sub&gt;_SINCE&lt;/sub&gt;</th>
<th>S&lt;sub&gt;_SINCE&lt;/sub&gt; WHERE STATUS_SINCE &lt; S&lt;sub&gt;_SINCE&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY ( S&lt;sub&gt;#&lt;/sub&gt; )</td>
<td>CONSTRAINT CR6 IS EMPTY</td>
</tr>
</tbody>
</table>

| SP<sub>_SINCE</sub> ( S<sub>#</sub>, P<sub>##</sub>, SINCE ) |
| KEY ( S<sub>#</sub> ) REFERENCES S<sub>_SINCE</sub> |

| FOREIGN KEY ( S<sub>#</sub> ) REFERENCES S<sub>_SINCE</sub> |
| CONSTRAINT CR9 IS EMPTY |

### Meeting the Nine Requirements (b): historical relvars only

| S<sub>_DURING</sub> ( S<sub>#</sub>, DURING ) |
| KEY ( S<sub>#</sub> ) REFERENCES S<sub>_DURING</sub> |

| USING ( DURING ) REFERENCES S<sub>_STATUS_DURING</sub> |
| FOREIGN KEY ( S<sub>#</sub>, DURING ) REFERENCES S<sub>_DURING</sub> |

| SP<sub>_DURING</sub> ( S<sub>#</sub>, P<sub>##</sub>, DURING ) |
| KEY ( S<sub>#</sub>, P<sub>##</sub>, DURING ) REFERENCES S<sub>_DURING</sub> |

| USING ( DURING ) REFERENCES S<sub>_DURING</sub> |

### SQL:2011's Counterpart of U_foreign key

CREATE TABLE S<sub>_STATUS_DURING</sub> ( S<sub>#</sub>, STATUS INTEGER, STATUS_FROM DATE, STATUS_TO DATE, PRIMARY KEY ( S<sub>#</sub>, PERIOD ), FOREIGN KEY ( S<sub>#</sub>, PERIOD ) REFERENCES S<sub>_DURING</sub> ( S<sub>#</sub>, PERIOD )

A foreign key thus specified is equivalent to a USING foreign key constraint in Tutorial D. Note that the referenced columns and PERIOD spec must be given.
Meeting the Nine Requirements (c): current and historical relvars

Very difficult, even with shorthands defined so far. E.g.,

Requirement R9:
If the database shows supplier Sx as being able to supply any part Py on day d, then it must also show supplier Sx as being under contract on day d.

\[
\text{CONSTRAINT BR9_A IS_EMPTY}
\]
\[
\text{CONSTRAINT BR9_B WITH ( EXTEND S_SINCE ;}
\]
\[
\text{(DURING := INTERVAL_DATE ([S#_SINCE : LAST_DATE ()])})
\]
\[
\text{SP_DURING (S#, DURING) AS T3 ;}
\]
\[
\text{USING (DURING) T3 \subseteq T2 \rangle (Note U_ form of relational comparison operator)}
\]

So, to cut a long story short:

\[
\text{VAR S_SINCE RELATION}
\]
\[
\{ S# S#, S#_SINCE DATE SINCE_FOR { S# } HISTORY_IN ( S_DURING ) ,
\]
\[
\text{STATUS INTEGER,}
\]
\[
\text{STATUS_SINCE DATE SINCE_FOR { STATUS } HISTORY_IN ( S_STATUS_DURING )}
\]
\[
\text{KEY ( S# ) ;}
\]
\[
\text{VAR SP_SINCE RELATION}
\]
\[
\{ S# S#, P# P# , SINCE DATE SINCE_FOR { S#, P# } HISTORY_IN ( SP_DURING )
\]
\[
\text{KEY ( S#, P# )}
\]
\[
\text{FOREIGN KEY ( S# ) REFERENCES S_SINCE ;}
\]

and we conjecture that the historical relvar definitions can be generated automatically.

Current and Historical Relvars in SQL:2011

SQL:2011 offers no special support for horizontal decomposition.

Database Queries

In Chapter 13, twelve generic queries of varying complexity are presented and then solved:
- a. for current relvars only
- b. for historical relvars only
- c. for both current and historical relvars

The c. section raises requirement for virtual relvars (views) that "undo" horizontal decomposition, such as:

\[
\text{VAR S_DURING_NOW_AND_THEN VIRTUAL}
\]
\[
\text{S_DURING UNION}
\]
\[
\text{(EXTEND S_SINCE ;}
\]
\[
\text{(DURING := INTERVAL_DATE ([ S#_SINCE : LAST_DATE () ]))}
\]
\[
\text{S#,DURING})
\]

Query Example

Example for c. (both current and historical relvars):

Get supplier numbers for suppliers who were able to supply both part P1 and part P2 at the same time

\[
\text{WITH ( EXTEND SP_SINCE :}
\]
\[
\text{(DURING := INTERVAL_DATE ([ SINCE : LAST_DATE () ]))}
\]
\[
\text{S#, P#, DURING) AS T1 ,}
\]
\[
\text{(SP_DURING UNION T1) AS T2 ,}
\]
\[
\text{(T2 WHERE P# = P#('P1')) ( S#, DURING) AS T3 ,}
\]
\[
\text{(T2 WHERE P# = P#('P2')) ( S#, DURING) AS T4 ,}
\]
\[
\text{(USING ( DURING) T3 JOIN T4}) AS T5 ;}
\]
\[
\text{T5 ( S# )}
\]
CHAPTER 14: Database Updates

What Are The Problems?

Thirteen generic update operations of varying complexity are presented in terms of addition, removal or replacement of propositions. E.g.:

Add the proposition "Supplier S2 was under contract from day 5 to day 6".

Remove the proposition "Supplier S1 was able to supply part P1 from day 5 to day 6".

Replace the proposition "Supplier S2 was able to supply part P1 from day 3 to day 4" by the proposition "Supplier S2 was able to supply part P1 from day 5 to day 7".

Inevitable conclusion is need for U_update operators ...

U_update operators

*U_INSERT*:

USING (ACL) \( \triangleright \) INSERT \( R \) \( r \) \( \triangleright \);

is shorthand for \( R := USING (ACL) \cup R \cup r \);

*U_DELETE*:

USING (ACL) \( \triangleright \) DELETE \( R \) \( WHERE \) \( p \) \( \triangleright \);

is shorthand for \( R := USING (ACL) \cup \{ \) WHERE NOT \( p \) \( \triangleright \}\);

and there's "U_UPDATE" too, of course (difficult to define formally)

But U_update operators aren't all that's needed ...

The PORTION Clause

<table>
<thead>
<tr>
<th>S#</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>[d03 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d02 : d05]</td>
</tr>
</tbody>
</table>

Replace the proposition "Supplier S1 was under contract from day 4 to day 8" by "Supplier S2 was under contract from day 6 to day 7".

(A trifle unreasonable but must be doable!)

We introduce PORTION:

UPDATE S_DURING WHERE S# = S# ( 'S1' )
PORTION ( DURING = INTERVAL_DATE ( [ d04 : d08 ] ) )
( S# = S# ( 'S2' ),
DURING = INTERVAL_DATE ( [ d06 : d07 ] ) ) ;

yielding:

<table>
<thead>
<tr>
<th>S#</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>[d03 : d03]</td>
</tr>
<tr>
<td>S1</td>
<td>[d09 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d02 : d07]</td>
</tr>
</tbody>
</table>

The Example Database

<table>
<thead>
<tr>
<th>S#</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>[d04 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>[d02 : d04]</td>
</tr>
<tr>
<td>S2</td>
<td>[d07 : d10]</td>
</tr>
<tr>
<td>S3</td>
<td>[d03 : d10]</td>
</tr>
<tr>
<td>S4</td>
<td>[d04 : d10]</td>
</tr>
<tr>
<td>S5</td>
<td>[d02 : d10]</td>
</tr>
</tbody>
</table>

Predicate: "Supplier S# was under contract throughout DURING (and not immediately before or after DURING)."

<table>
<thead>
<tr>
<th>S#</th>
<th>P#</th>
<th>DURING</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td>[d04 : d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>[d05 : d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
<td>[d06 : d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P4</td>
<td>[d07 : d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P5</td>
<td>[d08 : d10]</td>
</tr>
<tr>
<td>S1</td>
<td>P6</td>
<td>[d09 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P1</td>
<td>[d03 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P2</td>
<td>[d04 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P3</td>
<td>[d05 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P4</td>
<td>[d06 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P5</td>
<td>[d07 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P6</td>
<td>[d08 : d10]</td>
</tr>
<tr>
<td>S2</td>
<td>P7</td>
<td>[d09 : d10]</td>
</tr>
</tbody>
</table>

Predicate: "Supplier S# was able to supply part P# throughout DURING (and not immediately before or after DURING)."

Regular INSERT, UPDATE, DELETE become too difficult for many common purposes ...

U_update operators in SQL:2011

SQL:2011 has no counterparts of U_update operators.
### The PORTION Clause in SQL:2011

The PORTION Clause in SQL:2011 allows for selective updates and deletions based on specified portions of time.

**Example:**

**Update Clause:**

```
UPDATE S_DURING
FOR PORTION OF DURING FROM d06 TO d08
SET S# := S# ( 'S2' )
WHERE S# = S# ( 'S1' );
```

**Delete Clause:**

```
DELETE S_DURING
FOR PORTION OF DURING FROM d06 TO d08
WHERE S# = S# ( 'S1' );
```

### “Deleting a Portion” in SQL:2011

**Example:**

```
DELETE S_DURING
FOR PORTION OF DURING FROM d06 TO d08
WHERE S# = S# ( 'S1' );
```

### Updating the Combination View

Finally, we need to be able to apply update operators to the virtual relvar that combines current state with history. So we propose to add a COMBINED_IN specification to relvar declaration syntax, for that express purpose. E.g.:

```
VAR S_SINCE RELATION
{ S# S#, S#_SINCE DATE SINCE_FOR { S# }
  HISTORY_IN ( S_DURING ),
  STATUS INTEGER,
  STATUS_SINCE DATE SINCE_FOR { STATUS }
  HISTORY_IN ( S_STATUS_DURING ),
  COMBINED_IN ( S_DURING_NOW_AND_THEN )
  KEY { S# };
```

### Proposed Terminology

**Stated times** = "valid times"

**Logged times** = "transaction times"

**Justification for proposed terms:**

The stated times of proposition $p$ are times when, according to our current belief, $p$ was, is or will be true. The logged times of proposition $q$ are times (in the past and present only) when the database recorded $q$ as being true.

[If $q$ includes a stated time, then some might call "$q$ during logged time $[t_1:t_2]$" a "bitemporal proposition and hence talk about "bitemporal relations". We don't.]
Logged Times in SQL:2011

CREATE TABLE S_DURING
  ( S# S#, S#_FROM DATE, S#_TO DATE, SYS_FROM TIMESTAMP, SYS_TO TIMESTAMP, PERIOD FOR DURING (S#_FROM, S#_TO), PERIOD FOR SYSTEM_TIME (SYS_FROM, SYS_TO), PRIMARY KEY ( S#, DURING WITHOUT OVERLAPS ) )

WITH SYSTEM VERSIONING – optional extra;

No more than one system time period spec allowed.
Some people call this a “bitemporal table”.

“System Versioning” in SQL:2011

WITH SYSTEM VERSIONING implies:
• rows with end-of-time “to” system time values are current
• other rows are historical
• updates are applied to current rows only but result in new historical rows being inserted
• table referenced in FROM clause yields current rows only unless overridden by a FOR SYSTEM TIME specification
• e.g. FOR SYSTEM TIME FROM t1 TO t2 BETWEEN t1 AND t2 AS OF t

Chapter 16: Point Types Revisited

Detailed investigation of point types and the significance of scale (preferred term to “granularity”). Includes discussion of:

If point type pt2 is a proper subtype of pt1 (under specialisation by constraint), what are the consequences for types INTERVAL_pt2 and INTERVAL_pt1?
(E.g.: EVEN_INTEGER and INTEGER)

What about nonuniform scales, as with pH values, Richter values and prime numbers?

What about cyclic point types, such as WEEKDAY and times of day?
Consequences of a < b being equivalent to a ≠ b for all (a,b), leading to modified definitions of various interval operators.

Is there any point in considering continuous point types? We conclude not, because you lose some operators and gain none.

The End