A Point-based Temporal Extension of SQL

David Toman

BRICS

Basic Research in Computer Science,
Centre of the Danish National Research Foundation
University of Aarhus

and

Department of Computer Science
University of Toronto

List of Slides

Why yet another Temporal SQL?

Goals of the talk:

1. to show *problems* with the current approach(es), and
2. to propose a *simple* and *semantically sound* alternative.

Plan of the talk:

- Introduction: Temporal Databases and Intervals
- Inherent problems in current proposals
- An alternative: SQL/TP

  ⇒ Syntax, Semantics, Examples

  ⇒ Query Evaluation

The currently most prominent proposals of temporal extensions of SQL are based on TSQL2 derivatives.

This choice would “standardize” all problematic issues present in TSQL2’s semantics (e.g., representation-dependent queries).

Also such an extension introduces very cumbersome syntax—the constructs depend on the concrete encoding of sets of time instants.

Temporal Databases

Abstract Temporal Databases

- relational database *with an additional data type*:
  $\Rightarrow$ (discrete) linear order $(T, <)$ of *time instants*
- leads to very large (potentially infinite) relations

Concrete Temporal Databases

- a compact encoding of Abstract TDBs
  $\Rightarrow$ *semantics* given by mapping ($\|.\|$) to Abstract TDBs

The relation is equivalent to the previous graph and is infinite in general.

The abstract relations capture the content of the data.

Even if we insisted on closed periods in the graph a finer granularity would lead to much “bigger” relations.

We allow only finite duplication to make relational operations meaningful.

The time is integer-like with granularity 1 year.

Running Example

Works

Name	Company	Year
John	IBM	1990
John	IBM	1991
John	DEC	1992
John	Microsoft	1993
John	Microsoft	...
Mary	DEC	1989
Mary	DEC	1990
Mary	IBM	1993
Mary	IBM	...
Steve	IBM	1995
Steve	IBM	...

$\xleftarrow{\|\cdot\|}$

Works

Name	Company	Ymin – Ymax
John	IBM	1990 – 1991
John	DEC	1992 – 1992
John	MicroSoft	1993 – ∞
Mary	DEC	1989 – 1990
Mary	IBM	1993 – ∞
Steve	IBM	1995 – 1996
Steve	IBM	1997 – ∞

David Toman, BRICS

Abstract vs. Concrete terminology applies to tuples, relations, and databases in the expected way.

Intervals vs. True Intervals

- Intervals as encodings of sets of time instants:

```
king("Charles IV", "Czech Kingdom",[1347,1378])
king("Casimir III","Poland",       [1333,1370])
```

- Intervals as points in two-dimensional space:

```
electricity("Jones", 40, 05/15/96, 06/15/96)
electricity("Smith", 35, 05/01/96, 06/01/96)
```

David Toman, BRICS

The two data types have a completely different meaning; completely different set of operations (set operations only meaningful on the first approach).

Moral: don't mix apples and oranges!

Interval Encoding and Duplicates

- Finite *duplication*

  $\Rightarrow$ in the *Abstract* Temporal Relations

- Duplicate semantics *must be* respected by $\|.\|$

  $\Rightarrow$ $\{(a,[0,2]),(a,[1,3])\}$ and $\{(a,[0,3]),(a,[1,2])\}$
  represent the **same** multiset *with 6 elements*

  $$\{(a,0),(a,1),(a,1),(a,2),(a,2),(a,3)\};$$

  therefore $\|\{(a,[0,2]),(a,[1,3])\}\| = \|\{(a,[0,3]),(a,[1,2])\}\|$.

  $\Rightarrow$ counting number of intervals
  in the *encoding* makes little sense...

To achieve compatibility with SQL we need to handle duplicates.

A big problem for most current approaches (they hack it). Commonly badly matched with the intended semantics. Reason: the duplicates are introduced on the level of concrete relations rather than abstract relations.

Temporal Query Languages

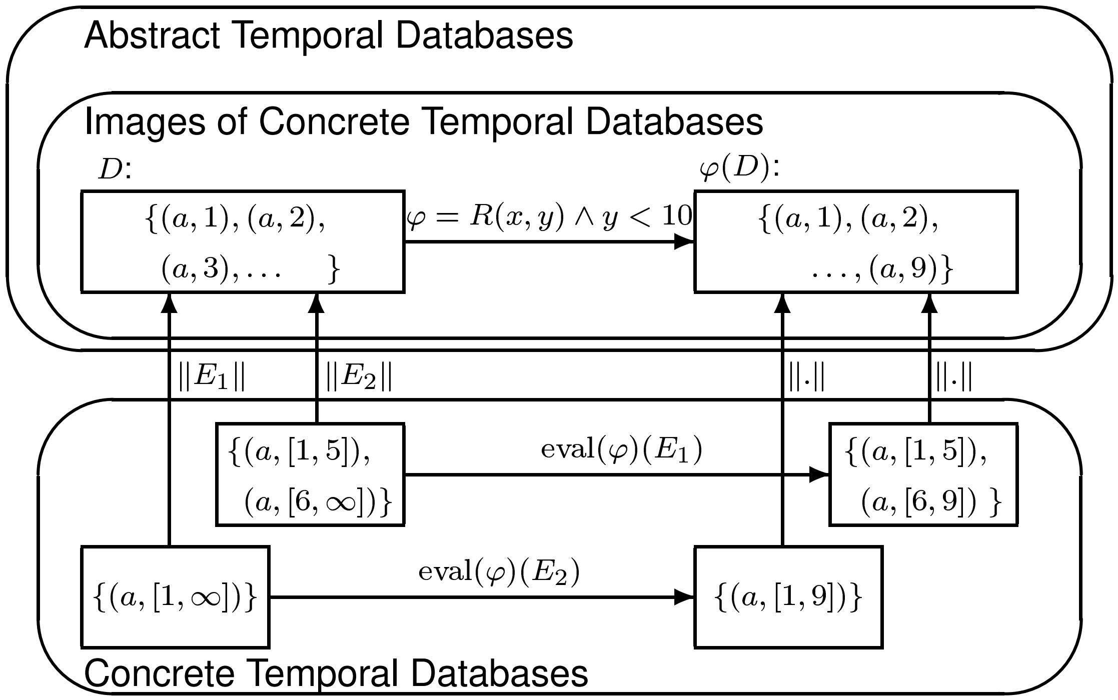
- Point-based:
  Two-sorted First Order Logic (Temporal RC)
  First Order Temporal Logic (FOTL)
  SQL/TP (with duplicates)
- Point-based, but with attributes ranging over intervals:
  TQuel [Snodgrass, 1987]
  TSQL/2 [Snodgrass, 1993]
  ATSQL [Böhlen and Jensen, 1996]
  SQL/Temporal [Snodgrass et al., 1996]
- Interval-based:
  SQL(/TP); intervals as pairs of points

The first group of languages defines its semantics over the abstract databases: clearly representation-independent (no encoding is involved), but not too practical: works over infinite or extremely large relations.

The second group defines its semantics over the concrete databases while trying to mimic a point based semantics over the meanings (not very successfully—we see examples later).

The third group is essentially a two-dimensional version of (1); anyway, we'll see that we can't restrict ourselves to a fixed dimension and thus (1) and (3) are essentially the same.

Interval Encoding



This is how the $\|.\|$ **has** to interact with queries so everything *fits* together:

1. top: abstract world of meanings and abstract query languages
2. bottom: concrete world of the encoding and concrete query languages

Interval valued Attributes

Example: “list all people who have a gap in their employment (with a single company)”

```
select  r.name
from    Works r, Works s
where   r.name=s.name
  and   r.company=s.company
  and   validtime(r) precedes validtime(s)
```

Does not work in general:

⇒ `Works` has to be coalesced and can not contain duplicates.

Definition: A temporal relation is *coalesced* if every fact is associated with maximal non-overlapping intervals.

Attempt1: ATSQL, SQL/Temporal (nonsequenced semantics; the only really temporal semantics of these two languages).

- essentially SQL over the concrete relations (with a lot of syntactic sugar),
- little help to write temporal queries that respect $\|.\|$.
- non generic queries!

In addition ATSQL, SQL/Temporal have other two (incompatible) semantics.

The example is simplistic, but not easy to fix (so the fix works for general queries).

Coalescing

Example: "list all people who have a gap in their employment"

```
select  r.name
from    Works r, Works s
where   r.name=s.name
  and   validtime(r) precedes validtime(s)
```

Again, this doesn't work:

`Works` has to be re-coalesced **after projecting out** company
$\Rightarrow$ can't be detected automaticaly
$\Rightarrow$ need for *explicit coalescing operator* or the use subqueries.

Attempt2: TSQL2, TQuel.

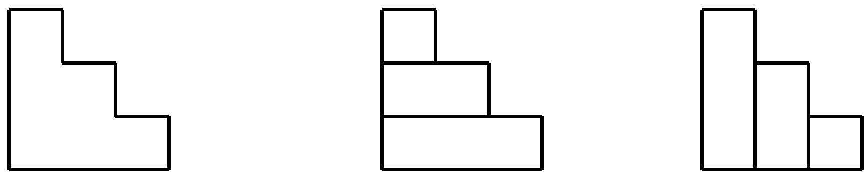
Recognizes the need to respect $\|.\|$. But:

- only 1 temporal dimension (will work with TL)
- coalescing is explicit (non declarative features) or implicit (unclear semantics)

Failure of Coalescing

Known facts:

- there is no unique coalescing for temporal dimension > 1



- FO-complete query languages can not avoid arbitrarily large dimensionality during query evaluation [Abiteboul et al., 1996, Toman and Niwinski, 1996]

$\Rightarrow$ non-$\|.\|$-generic queries

Coalescing *cannot* work in general.

Temporal elements? No:

- still can write representation dependent queries
- no fixed-size tuples
- what are the operations?
  how many? (points: 3, intervals: 13, bitemporal elements: ?)

Folding and Unfolding

Definition Let R be a concrete temporal relation. Then

$$\mathrm{unfold}(R) = \{(t,x) : \exists I.R(I,x) \wedge t \in I\}.$$

IXRM [Lorentzos, 1993]

- interval, multidimensional approach
- unfold

  $\Rightarrow$ EXPSPACE for finite intervals

  $\Rightarrow$ not defined for unbounded intervals
- cannot be used as a basis for *efficient* query evaluation

May work for defining semantics (but that can be done easier).

Hopeless for query evaluation.

What is the Problem?

- Intended **semantics** is based on

  Abstract Temporal Databases

- Actual **syntax** is defined with respect to the

  Concrete Encoding (Intervals)

- Consequences:
  1. un-intuitive and non-declarative syntax:
     $\Rightarrow$ explicit non-logical operators (coalescing)
     $\Rightarrow$ usual logical equivalences do not hold
  2. semantic problems:
     $\Rightarrow$ non $\|.\|$-generic queries
  3. unclear/unknown expressive power

the syntax/semantics "split personality" of the languages is the cause for most of the problems.

The Solution: SQL/TP

Idea: Temporal attributes range over individual time instants

- Syntax and Semantics:
  ⇒ *exactly like* extending SQL with a new data type
  ⇒ support for finite duplication
- Concrete (physical) representation:
  ⇒ compact representation using intervals
  ⇒ syntactic restrictions to maintain closure (safety)
- Query evaluation:
  ⇒ efficiency: depends on the (size of the) encoding only
  ⇒ allows compilation to SQL/92

The goals:

(1) The syntax and semantics of standard SQL has to be extended in a *natural way*: by adding a new data type `time` that behaves (almost) identically as the existing data types, and

(2) The syntax and semantics are *independent* of the chosen encoding of timestamps.

The chosen language is derived from calculus with finite duplicates.

The choice of encoding is not critical: we aim on subsuming TSQL, so we have chosen TSQL's encoding of temporal databases.

Efficient of query evaluation means that it uses only elements in the active domain of the encoding (plus some small neighborhoods of `Time` values).

SQL/TP: syntax

```
<exp> ::= <ddl> | <dql> | <dml>

<ddl> ::= create table <rid> ( <signat> ) |
          create view  <rid> ( <dql> )

<dql> ::= select [all] <slist> [from <flist>]
                 [where <cond>] [group by <glist>] |
          ( <dql> ) union [all] ( <dql> ) |
          ( <dql> ) except [all] ( <dql> ) |
          ( <dql> ) intersect [all] ( <dql> )

<dml> ::= insert [all] into <rid> ( <dql> ) |
          delete [all] from <rid> ( <dql> )
```

David Toman, BRICS

Standard SQL, all nesting of queries is realized in the `from` clause.

The `having` clause and nesting in the `where` clause are easily definable using nesting in the `from` clause and thus are considered to be just a syntactic sugar.

Where is the “temporal extension”? In the data definition language: when we define an attribute of type `time` (next slide)...

SQL/TP: Encoding of Time

- additional data type for *time*

  `<type>    ::= ... | time [using <enc>] | ...`

- with *encoding modifiers* **`<enc>`**:

  ⇒ **`points`**

  ⇒ **`bounded intervals`**

  ⇒ **`unbounded intervals`**

Other classes of constraints can be readily incorporated

`points` is for atomic events (e.g., transaction time).

`bounded intervals` is for facts with bounded “duration”.

`unbounded intervals` is for facts with unbounded duration (e.g., x holds from now on)

Easy to introduce other constraint classes (back-door for the CDB folks): If you can’t beat them (SQL), join them.

Examples

"List all years when nobody worked for IBM"

```
( select t as year from true ) except
( select year from Works where company='IBM' )
```

Result:

```
     ymin         ymax
======================
-infinity         1990
     1992         1992
```


The query can be answered by a complementation of the remainder of the projection on `year` from the `Works` relation. Note the use of the `true` pseudo-relation to achieve signature compatibility.

Note that the results of the queries are *concrete* temporal relations.

Extremely hard in TSQL (go and try, if you don't believe me!)

SQL/TP: features

- representation independence
- simple syntax and semantics
- first-order complete
- efficient query evaluation for interval-based encoding

- can express all $\|.\|$-generic SQL/Temporal queries
- (almost) fulfils migration requirements (ISO/ANSI)
- easily extensible to other encodings

- ...

Semantics is defined on the abstract relations.

Syntax and semantics are taken straight from SQL.

Allows execution of non-temporal queries at the current point or as LISP's "map" for every time instant.

The same query in SQL

```
select r1.ymax+1, r2.ymin-1
from   Works r1, Works r2
where  r1.company='IBM' and r2.company='IBM'
  and  r1.ymax+1 <= r2.ymin-1
  and  not exists
  ( select *
    from   Works r3
    where  r3.company='IBM'
      and  ( ( r1.ymax < r3.ymin and
               r3.ymin < r2.ymin )
          or ( r1.ymax < r3.ymax and
               r3.ymax < r2.ymin ) ) )
```

⇒ this is what you write in (*nonsequenced*) SQL/Temporal.

Sequenced semantics would help with this query but any other *temporal* query in ATSQL or SQL/Temporal will look like this (just try to write a "A since B" query (even with use of coalescing; no luck? look at [Böhlen et al., 1996]).

Examples (cont.)

“List all people with gap in their job history”

```
select  r1.name
from    Works r1, notWorks r2, Works r3
where   r1.name = r2.name and r2.name = r3.name
  and   r1.year < r2.year and r2.year < r3.year
```

Result:

```
name
====================
Mary
```


This query is a three-way self join, where one of the tables is complemented. Cf. the slides in the beginning of the talk.

See discussion of this query in TSQL2 and other coalescing-based temporal query languages (transparencies 8 and 9).

Also, it naturally corresponds to the original calculus formula.

Examples (cont.)

“For every year list the number of employed people”

```
select year, count(name) from Works group by year
```

Result:

ymin	ymax	count
1989	1989	1
1990	1990	2
1991	1991	1
1992	1992	1
1993	1994	2
1995	1996	3
1997	infinity	3

Grouping by the temporal dimension is much harder: it is not clear, how this result can be directly obtained in TSQL2.

Coalescing does not help to answer this query: the input is coalesced and re-coalescing after projecting out the `name` attribute would erase the needed duplicates. This is the crucial difference between coalescing and our normalization procedure.

SQL/TP: Query Processing

Defined by compilation of the **whole** language to SQL/92.

$\Rightarrow$ compile : SQL/TP $\longrightarrow$ SQL/92

$$\begin{tikzcd}
\boxed{\text{Abstract TDB}} \arrow[d, "Q_{\text{SQL/TP}}"] & \boxed{\text{Concrete TDB}} \arrow[l, "\|\cdot\|"'] \arrow[d, "Q'_{\text{SQL/92}} = \text{compile}(Q)"] & \boxed{\text{Physical DB}} \arrow[l] \arrow[d, "\text{eval}(Q')"] \\
\boxed{\text{Abstract Relation}} & \boxed{\text{Concrete Relation}} \arrow[l, "\|\cdot\|"'] & \boxed{\text{Physical Relation}} \arrow[l]
\end{tikzcd}$$

David Toman, BRICS

So far we only hinted about the query evaluation of SQL/TP queries over the concrete databases (note that the semantics is defined over the abstract databases.

We chose compilation: easy implementation on top of an existing RDBMS.

Also we identify operations that we may want to add to a RDBMS to facilitate more efficient evaluation of temporal queries.

SQL/TP: Query Processing (cont.)

DDL: trivial *extension* of the schema

DQL: two goals

1. elimination of point-valued temporal attributes:
   * $Q^{\text{SQL/TP}}(t,x) \iff \exists I.Q^{\text{SQL/92}}(I,x)[t \in I]$
   * quantifier elimination for linear orders

     details [Toman, 1996]
2. simulation of set operations in SQL/92
   * a *normalization* technique

DML: similarly to DQL.

While the top level temporal attributes have to be independent, this may not be true for all the subqueries. The conditional formulas allow us to propagate and eventually eliminate such dependencies (they are constraints "global" for the whole subquery: in a CDB approach such a constraint would be attached to every tuple in the answer to the subquery).

Select Block

```
select r.name, s.name, r.year
from   Works r, Works s
where  r.year=s.year
```

John	1992	1992

$\times$

Mary	1989	1990
Steve	1990	1993

$\Downarrow$

John	1992	1992	Mary	1989	1990
John	1992	1992	Steve	1990	1993

$\Downarrow \sigma(\texttt{r.ymin} \leq \texttt{s.ymax}, \texttt{r.ymin} \leq \texttt{r.ymax},)$

John	1992	1992	Steve	1992	1992

$[\texttt{r.year} = \texttt{s.year}]$

Select block is translated in two steps: first we translate the product–selection part (cf. query in Example 1).

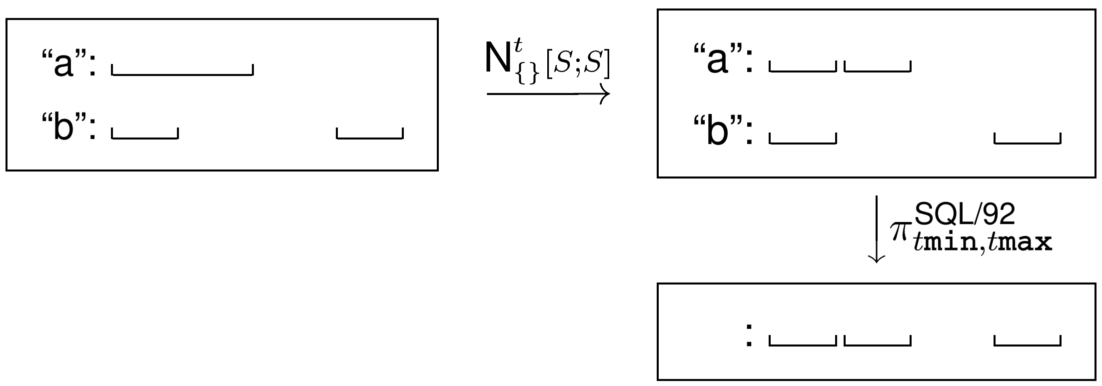
The selection condition on the temporal attributes is derived using quantifier elimination of the point attributes from

$$\exists t_1, t_2 . t_1\texttt{min} \leq t_1 \leq t_1\texttt{max} \wedge t_2\texttt{min} \leq t_2 \leq t_2\texttt{max} \wedge t_1 = t_2$$

The input tables may be results to subqueries: we need to add the conditional formulas associated with these subqueries.

We derive a resulting condition by eliminating all but the point temporal attributes (in this case $t_1 = t_2$).

Select Block (duplicate elimination)



⇒ Aggregation works similarly.

The second step is the projection, duplicate elimination (and aggregation, if needed).

The top left and right tables are $\|.\|$-equivalent.

The standard projection with duplicate elimination applied on the top right table produces the desired result.

The normalization N guarantees that the set/bag operation of SQL (e.g. projection with duplicate elimination) works correctly.

Time-compatible Queries

Definition. $Q_1, \ldots, Q_k$ queries with compatible signatures, X a subset of their data attributes, and t a temporal attribute.

$Q_1, \ldots, Q_k$ are *t-compatible on X* if

$$a, b \in \bigcup Q_i(D) \text{ such that } \pi_X(a) = \pi_X(b)$$
$$\text{then } \pi_{\{t\}}(\|a\|) \text{ and } \pi_{\{t\}}(\|b\|) \text{ are identical or disjoint.}$$

$Q_1, \ldots, Q_k$ are *time-compatible* on X if $Q_1, \ldots, Q_k$ are t-compatible on X for all temporal attributes t in the signature.

Lemma. There are first order queries $\mathsf{N}^t_X[Q_i; Q_1, \ldots, Q_k]$ s.t.

(1) $\|Q_i(D)\| = \|\,\mathsf{N}^t_X[Q_i; Q_1, \ldots, Q_k](D)\|$
(2) $\{\mathsf{N}^t_X[Q_i; Q_1, \ldots, Q_k] : 0 < i \leq k\}$ are t-compatible

David Toman, BRICS

The t-compatibility guarantees that the interval values of t behave like points: they are identical when the representation is identical and disjoint when the representation differs.

This approach can (and is) used for all set/bag operations. It also allows the alternative use of merge-joins (cf. intersection) in the `from` clause (this is too tricky to be found by the underlying DBMS).

Set Operations

Q_1 `<setop>` Q_2 is translated using the following Lemma:

Lemma. Let $\{Q_1, Q_2\}$ be time-compatible. Then

$$\|Q_1\| \;\; \mathtt{op} \;\; \|Q_2\| = \|Q_1 \;\; \mathtt{op} \;\; Q_2\|$$

where $\mathtt{op} \in \{\cap, \cup, -\}$ is a set operator on relations.

$\Rightarrow$ compile$(Q_1 \;\; \mathtt{op} \;\; Q_2) =$ N$[Q_1; Q_1, Q_2] \;\; \mathtt{op} \;\;$ N$[Q_2; Q_1, Q_2]$
$\Rightarrow$ in $O(n \log n)$ of the combined size of Q_1 and Q_2

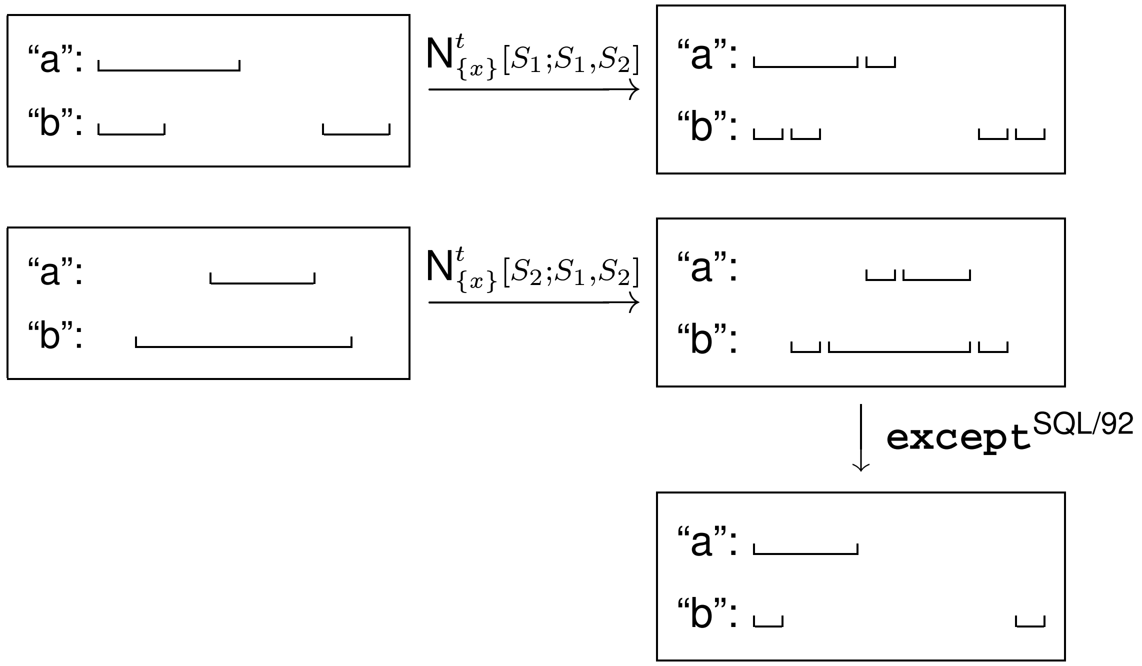
The set operations are based on N similarly to duplicate elimination.

N not necessary in the case of additive union (`union all`).

The intersection technique can be used for merge-joins in the translations of the select block.

N can be based on sorting: the cost may amortize in subsequent join and duplicate elimination steps.

Set Operations (cont.)



This example shows how a difference is computed using N.

Note that it preserves duplicates, if necessary.

Set Operations (cont.)

$$\{Q_i[\varphi_i]\}\ \texttt{union}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i[\varphi_i \wedge \neg \textstyle\bigwedge_j \psi_j], R_j[\psi_j \wedge \neg \textstyle\bigwedge_i \varphi_i], Q_i \cup R_j[\varphi_i \wedge \psi_j]\}$$

$$\{Q_i[\varphi_i]\}\ \texttt{except}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i[\varphi_i \wedge \neg \textstyle\bigwedge_j \psi_j], Q_i - R_j[\varphi_i \wedge \psi_j]\}$$

$$\{Q_i[\varphi_i]\}\ \texttt{intersect}\ \{R_j[\psi_j]\} \mapsto$$
$$\{Q_i \cap R_j[\varphi_i \wedge \psi_j]\}$$

The conditional queries are handled using the above rules: the goal is to push the set operation towards the unconditional parts of the queries.

All the conditional formulas are computed at compile time.

This transformation is at most quadratic (not exponential).

Future Work

- Query optimization

  $\Rightarrow$ Algebraic laws for N
  we know it is idempotent, . . .

- More complex temporal domains
- More expressive encoding
- Updates in place
- N as SQL expression or native implementation?

Conclusion

SQL/TP: A new approach to temporal extensions of SQL.

Features:

1. Clean syntax and semantics (with finite duplicates).
2. Efficient implementation.
3. Translation to SQL/92.
4. Localization of temporal extensions to a single operation N.

Database vendors ought to love it, especially points (3) and (4).

slides: `http://www.cs.toronto.edu/~david/slides-dood97.ps.Z`.

References

[Abiteboul et al., 1996] Abiteboul, S., Herr, L., and den Bussche, J. V. (1996). Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, Montréal, Canada.

[Böhlen et al., 1996] Böhlen, M., Chomicki, J., Snodgrass, R., and Toman, D. (1996). Querying TSQL2 Databases with Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France. Springer Verlag, LNCS 1057.

[Böhlen and Jensen, 1996] Böhlen, M. H. and Jensen, C. S. (1996). Seamless integration of time into sql. Technical Report R-96-2094, Institute for Electronic Systems, Aalborg University.

[Lorentzos, 1993] Lorentzos, N. A. (1993). The Interval-extended Relational Model and Its Application to Valid-time Databases. In Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., editors, *Temporal Databases: Theory, Design, and Implementation*, pages 67–91. Benjamin/Cummings.

[Snodgrass, 1987] Snodgrass, R. (1987). The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298.

[Snodgrass, 1993] Snodgrass, R. T. (1993). *An Overview of TQuel*, chapter 6, pages 141–182. Benjamin/Cummings.

[Snodgrass et al., 1996] Snodgrass, R. T., Böhlen, M. H., Jensen, C. S., and Steiner, A. (1996). Adding valid time to sql/temporal. ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2 21/11/96, (change proposal), International Organization for Standardization.

[Toman, 1996] Toman, D. (1996). Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems*, Montréal, Canada.

[Toman and Niwinski, 1996] Toman, D. and Niwinski, D. (1996). First-order queries over temporal databases inexpressible in temporal logic. In *International Conference on Extending Database Technology*, Avignon, France.