

Temporal Statement Modifiers

David Caspar

May 8, 2010

Temporal statement modifiers

Temporal statement modifiers are an approach how to extend a given non-temporal query language such that the extended language provides temporal semantics.

Böhlen, M. H., Jensen, C. S., and Snodgrass, R. T. Temporal statement modifiers. *ACM Trans. Database Syst.* 25, 4 (2000), 407-456.

Why temporal semantics?

Consider the following non-temporal relation `Employee`:

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

Why temporal semantics?

Consider the following non-temporal relation `Employee`:

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

For every year captured in this relation, we want to know how many employees our company had:

Employees	VTIME
2	1999 - 2000
3	2001 - 2002
2	2003 - 2004
1	2005 - 2006

Expressing this query in SQL is difficult...

Why temporal semantics?

The same query, in temporally extended SQL:

```
SEQ VT
  SELECT  COUNT(*) AS "Employees"
  FROM Employee
```

Why temporal semantics?

The same query, in temporally extended SQL:

```
SEQ VT
SELECT COUNT(*) AS "Employees"
FROM Employee
```

Evaluated on our relation...

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

...we get the desired result:

Employees	VTIME
2	1999 - 2000
3	2001 - 2002
2	2003 - 2004
1	2005 - 2006

Snapshots

A temporal relation

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

Snapshots

A temporal relation

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

can also be seen as a sequence of non-temporal snapshots:

1999			2000			2001		
Name	Salary		Name	Salary		Name	Salary	
Paul	20000	→	Paul	20000	→	Paul	20000	→ ...
Sarah	30000		Sarah	30000		Sarah	30000	
						Peter	20000	

Keeping existing non-temporal queries and databases

When migrating huge databases, we usually cannot afford it to migrate them all to the new temporal data model.

Keeping existing non-temporal queries and databases

When migrating huge databases, we usually cannot afford it to migrate them all to the new temporal data model.

A temporal data model must ensure that existing non-temporal databases and queries stay valid and continue to function properly.

Formal definition (upward compatibility):

Let $M^x = (DS^x, QL^x)$ and $M^y = (DS^y, QL^y)$ be two data models.
 M^y is upward compatible with M^x iff

- ▶ $\forall db^x \in DS^x : db^x \in DS^y$
- ▶ $\forall s^x \in QL^x : s^x \in QL^y$
- ▶ $\forall db^x \in DS^x, \forall s^x \in QL^x : [s^x(db^x)]_{M^x} = [s^x(db^x)]_{M^y}$

Keeping existing non-temporal queries, cont'd

Suppose we converted a non-temporal relation to a temporal one...
...but we still have old, non-temporal queries working with it.

Problem:

How do we ensure non-temporal queries still function properly?

Keeping existing non-temporal queries, cont'd

Suppose we converted a non-temporal relation to a temporal one...
...but we still have old, non-temporal queries working with it.

Problem:

How do we ensure non-temporal queries still function properly?

Solution (temporal upward compatibility):

Non-temporal queries on temporal relations are evaluated on the current snapshots of the relations.

Keeping existing non-temporal queries

Example: Suppose we're at time point 2006. The query...

```
SELECT  COUNT(*) AS "Employees"  
FROM    Employee
```

...evaluated on Employee...

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

...respectively its current snapshot...

2006	
Name	Salary
Peter	25000

...produces the result:

Employees
1

Semantics of non-temporal query statements

A non-temporal query is always evaluated on the current snapshot of a relation.

The current snapshot of a non-temporal relation is the relation itself.

On the current snapshots, the semantics of non-temporal queries are those already known.

Temporal queries

Now, lets have a look at temporal queries like the following:

```
SEQ VT
  SELECT  COUNT(*) AS "Employees"
  FROM Employee
```

Temporal queries

Now, lets have a look at temporal queries like the following:

```
SEQ VT
  SELECT COUNT(*) AS "Employees"
  FROM Employee
```

For every non-temporal query q , there exists a corresponding temporal query q_t , such that

- **Syntax:** q_t is written as

$$q_t = S_1 q S_2$$

where S_1 and S_2 are strings, called statement modifiers

- **Semantics:** Taking a snapshot at time t of q_t 's result is equivalent to evaluating q on a snapshot of the argument relation at t .

Temporal queries - Example

The query

```
SEQ VT SELECT COUNT(*) AS "Employees"  
FROM Employee
```

evaluated on Employee:

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

Step 1a: Take the final result...

Employees	VTIME
2	1999 - 2000
3	2001 - 2002
2	2003 - 2004
1	2005 - 2006

Step 1b: Take snapshot of
Employee at 1999:

Name	Salary
Paul	20000
Sarah	30000

Step 2a: Take snapshot at 1999

Step 2b: Compute result at snapshot

Employees
2

Sequenced semantics

The following query, evaluated on the relation Employee...

```
SEQ VT SELECT  COUNT(*) AS "Employees"  
FROM Employee
```



Employee

...is reduced to Employee's (non-temporal) snapshots:

```
SELECT  COUNT(*) AS "Employees"  
FROM Employee
```



... → Snapshot t → ...

But wait ...

Suppose we extended our query:

```
SEQ VT SELECT  COUNT(*) AS "Employees"  
  FROM Employee  
  WHERE VTIME(Employee)  
         OVERLAPS PERIOD '2000-2002'
```

But wait ...

Suppose we extended our query:

```
SEQ VT SELECT  COUNT(*) AS "Employees"  
FROM Employee  
WHERE VTIME(Employee)  
      OVERLAPS PERIOD '2000-2002'
```

The statement `VTIME(Employee)` cannot be evaluated based on single snapshots...

In this case, we can no longer think in terms of isolated snapshots.

But wait ...

Instead, we must also think of our temporal relation as a whole.

```
SEQ VT SELECT COUNT(*) AS "Employees"  
FROM Employee  
WHERE VTIME(Employee)  
CONTAINS '2000-2002'
```

Employee:

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

Result:

Employees	VTIME
2	1999 - 2002
1	2003 - 2004

Lineage Preservation

How should result tuples be coalesced ?

Lineage Preservation

How should result tuples be coalesced ?

Preserve timestamps of argument tuples.

Observations:

- ▶ Not every tuple in a argument relation is necessary to produce a specific result tuple.
- ▶ We are interested in those argument tuples that are really required and output the result over the smallest time period.
- ▶ Result tuples derived from such minimal argument tuples should be coalesced.

Lineage Preservation - Example

Recall our example query

```
SEQ VT SELECT COUNT(*) AS "Employees"  
FROM Employee
```

This time, we evaluated the query on...

Name	Salary	VTIME
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006
Lisa	25000	2007 - 2009

...and we get the following result:

Employees	VTIME
1	1999 - 2000
2	2001 - 2004
1	2005 - 2009

Is this query lineage preserving?

Lineage Preservation - Example

No!

But, if we split up the result tuple over the period 2005 - 2009,

Employees	VTIME
1	1999 - 2000
2	2001 - 2004
1	2005 - 2006
1	2007 - 2009

the query becomes lineage preserving, since it preserves much better the valid-time of the argument tuples.

Name	Salary	VTIME
Peter	25000	2003 - 2006
Lisa	25000	2007 - 2009

Valid time is just an attribute...

Temporal semantics is a great thing, but we want to use time stamps also in non-sequenced queries.

Valid time is just an attribute...

Temporal semantics is a great thing, but we want to use time stamps also in non-sequenced queries.

Non-restrictiveness

It should be possible to manipulate time stamps explicitly.

The keyword `NSEQ` denotes a query without sequenced semantics. Instead, timestamps are turned into regular attributes and non-temporal semantics is used.

Valid time is just an attribute...

Consider the following query:

```
NSEQ VT SELECT COUNT(*) AS "Employees"  
FROM Employee  
WHERE VTIME(Employee)  
CONTAINS '2000-2002'
```

If we evaluate this query on our
relation Employee...

Name	Salary	VTIME
Paul	20000	1999 - 2002
Sarah	30000	1999 - 2004
Peter	20000	2001 - 2002
Peter	25000	2003 - 2006

...we get the following non-temporal
result:

Employees
2

Temporal Statement Modifiers - Conclusion I

- ▶ Upward compatibility is maintained
 - ▶ Non-temporal relations are evaluated on the current snapshots of argument relations
 - ▶ On snapshots, semantics are the same as SQL
- ▶ Sequenced semantics is provided
 - ▶ Temporal queries take all snapshots of argument relations into account
 - ▶ Statement modifier SEQ
 - ▶ Require new semantics

Temporal Statement Modifiers - Conclusion II

- ▶ Lineage is preserved
 - ▶ Timestamps of result tuples are as similar to timestamps of required argument tuples as possible.
- ▶ Non-sequenced queries over timestamps are possible
 - ▶ Provides explicit controll over timestamps
 - ▶ Statement modifier NSEQ
 - ▶ Semantics like non-temporal SQL