Temporal and Spatial Data Management
Fall 2017

Sequenced Semantics
SL04

- Elements of the sequenced semantics
- Integration of the sequenced semantics into the DBMS kernel
- SQL implementation

Goal of Reduction to Snapshots

- There is a close relationship between a temporal and a nontemporal database:
  - the snapshot of a temporal relation at a time $t$ is a nontemporal relation.
- All nontemporal statements can be evaluated at each snapshot of a temporal database ("at each time point")
- There should be a close relationship between a temporal and a nontemporal statement:
  - a temporal aggregation should resemble a nontemporal aggregation.
- With SQL this is not the case (remember temporal join versus join).

Setup and Notations

- Relation schema: $R(A_1,...,A_n, T_S, T_E)$
- $A_1,...,A_n$ are the explicit attributes
- $T_S, T_E$ are temporal attributes
  - $T_S$: valid time start
  - $T_E$: valid time end
- $z^{n+2}$ denotes a tuple of arity $n + 2$
- We assume half open intervals: $[T_S, T_E)$
- We write $T$ to refer to the period $[T_S, T_E)$
- $t \in T \equiv T_S \leq t < T_E$
Reducibility of Temporal Operators

Snapshot Reducibility

- The timeslice operator maps a temporal to a nontemporal relation.
- Definition of the timeslice operator:

\[ \tau_t(r) = \{ z^n \mid \exists x \in r(z.A = x.A \land x.T_S \leq t < x.T_E) \} \]

- Two temporal relations, \( r \) and \( s \), are snapshot equivalent, \( r \equiv s \), iff for all times \( t \) their snapshots are identical.
- Definition of snapshot equivalence:

\[ r \equiv s \iff \forall t(\tau_t(r) = \tau_t(s)) \]

Example of Snapshot Equivalence

- Example: Two-day and four-day checkouts in the video example

<table>
<thead>
<tr>
<th>CustID</th>
<th>TapeNum</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C102</td>
<td>T1245</td>
<td>[19]</td>
</tr>
<tr>
<td>C102</td>
<td>T1245</td>
<td>[20]</td>
</tr>
</tbody>
</table>

- \( \tau_9(\text{checkout1}) = \{(C102, T1245)\} \)
- \( \tau_9(\text{checkout2}) = \{(C102, T1245)\} \)
- \( \tau_9(\text{checkout1}) = \{(C102, T1245)\} \)
- \( \tau_9(\text{checkout2}) = \{(C102, T1245)\} \)

- Apply the timeslice operator at each time point:

- checkout1 and checkout2 are snapshot equivalent.
- checkout1 and checkout2 are syntactically different.

Permission for non-commercial use is granted by the UZH Library. The UZH Library reserves the right to revoke this permission at any time. For any use beyond the granted permission, contact the UZH Library.
**Data Lineage/1**

The **lineage set**, \( L[\psi^T(R_1, \ldots, R_n)](z, t) \), of result tuple \( z \) at time \( t \) is the set of witness lists of argument tuples, \( \{r_1, \ldots, r_n\} \), \( r_i \in R_i \), from which \( z \) is derived:

\[
L[\sigma^T_\theta(R)](z, t) = \{(r) \mid r \in R \land z.A = r.A \land t \in r.T\}
\]

\[
L[\pi^T_B(R)](z, t) = \{(r) \mid r \in R \land z.B = r.B \land t \in r.T\}
\]

\[
L[\beta^T_F(R)](z, t) = \{(r) \mid r \in R \land z.B = r.B \land t \in r.T\}
\]

\[
L[R \cdot \top S](z, t) = \{(r, \bot) \mid r \in R \land z.A = r.A \land t \in r.T\}
\]

\[
L[R \cup \top S](z, t) = \{(r, \bot) \mid r \in R \land z.A = r.A \land t \in r.T\} \cup \{(\bot, s) \mid s \in S \land z.A = s.A \land t \in s.T\}
\]

\[
L[R \times \top S](z, t) = \{(r, s) \mid r \in R \land z.A = r.A \land t \in r.T \land s \in S \land z.C = s.C \land t \in s.T\}
\]

\[
L[R \triangleright \top S](z, t) = \{(r, \bot) \mid r \in R \land z.A = r.A \land t \in r.T\}
\]

**Change Preservation**

We use lineage to preserve changes (start and end of intervals).

- \( \{DB, 5K, [Feb, Jul]\} \neq \{DB, 5K, [Feb, Apr]\} \) \( \& \) \( \{DB, 5K, [Apr, Jul]\} \)
- scaling of values (based on old and new timestamps) gets possible

\( \psi^T \) is change preserving iff \( \forall z, z' \in \psi^T(R_1, \ldots, R_n) \):

\[
\forall t, t' \in z.T (L(z, t) = L(z, t')) \land \\
(z.T_S - 1 \in z'.T \Rightarrow L(z', z.T_S - 1) \neq L(z, z.T_S)) \land \\
(z.T_E \in z'.T \Rightarrow L(z', z.T_E) \neq L(z, z.T_S))
\]

Change preservation permits facts that hold for an entire interval but not a subinterval. Intervals are not coalesced automatically.

- \( \pi^T_D \)

**Extended Snapshot Reducibility**

We want to access time intervals in snapshot reducible operators.

- at each time point join long and short projects

\( \psi^T \) is extended snapshot reducible iff for all \( t \):

\[
\tau_t(\psi^T(R_1, \ldots, R_n)) = \pi_E(\tau_t(\epsilon U_1(R_1)), \ldots, \tau_t(\epsilon U_n(R_n)))
\]

For \( \psi \in \{\otimes, \sigma, \pi, \times, \square, \triangleright, \top\} \), \( E = \) schema of \( \psi^T(R_1, \ldots, R_n) \).

Extended snapshot reducibility supports snapshot reducibility and access to timestamps (by propagating timestamps with the extend operator \( \epsilon \)).

- at each snapshot reducibility requires projects: \( D^\psi AVG(DUR(T))(R) \)
Scaling/1

Consider a project relation $p$ that tracks project fundings $B$:

<table>
<thead>
<tr>
<th>$D$</th>
<th>$N$</th>
<th>$B$</th>
<th>$T_S$</th>
<th>$T_E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>1</td>
<td>181K</td>
<td>2013/2/1</td>
<td>2013/8/1</td>
</tr>
<tr>
<td>DB</td>
<td>2</td>
<td>196K</td>
<td>2013/5/1</td>
<td>2014/1/1</td>
</tr>
<tr>
<td>AI</td>
<td>1</td>
<td>153K</td>
<td>2013/4/1</td>
<td>2013/9/1</td>
</tr>
<tr>
<td>AI</td>
<td>2</td>
<td>120K</td>
<td>2013/4/1</td>
<td>2013/9/1</td>
</tr>
</tbody>
</table>

When aggregating such data according to snapshot reducibility we might want to scale the funding to the new time periods:

<table>
<thead>
<tr>
<th>$D$</th>
<th>$B$</th>
<th>$T_S$</th>
<th>$T_E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>89K</td>
<td>2013/2/1</td>
<td>2013/5/1</td>
</tr>
<tr>
<td>DB</td>
<td>165.6K</td>
<td>2013/5/1</td>
<td>2013/8/1</td>
</tr>
<tr>
<td>DB</td>
<td>122.4K</td>
<td>2013/8/1</td>
<td>2014/1/1</td>
</tr>
<tr>
<td>AI</td>
<td>273K</td>
<td>2013/4/1</td>
<td>2013/9/1</td>
</tr>
</tbody>
</table>

Scaling/2

Let $x$ be an attribute value to be scaled and $T_N$ and $T_O$ be interval timestamps such that $T_N \subseteq T_O$.

A scaling function $scale$ defines a weight $0 < w(T_N, T_O) \leq 1$ and scales $x$ accordingly:

$$scale(x, T_N, T_O) = x \cdot w(T_N, T_O), \text{ where } 0 < w(T_N, T_O) \leq 1$$

Lemma: Scaling must be a parameter of temporal operators, since for some temporal operators (e.g., aggregation, difference), it cannot be performed in a pre- or post-processing step.

The reason is that neither before nor after the temporal operator all the required attributes for scaling are available.

Table of Contents

- Elements of the sequenced semantics
  - Snapshot reducibility
  - Change preservation
  - Extended snapshot reducibility
  - Scaling
- Integration of temporal support in DBMS kernel
  - Algebraic adjustment operators
  - Reduction rules
  - SQL implementation

Integration of the Sequenced Semantics into the DBMS Kernel

- Goal:
  - Determine and implement in the DBMS kernel the functionality that is required to offer support for the sequenced semantics (snapshot reducibility + change preservation + extended snapshot reducibility + scaling).
  - The sequenced semantics supports all features that have been identified as important for processing time periods.

- Solution:
  - New algebraic adjustment operators that split periods into pieces, such that snapshot reducible queries only need equality for comparison.
  - Possibility to propagate and access the original timestamps.
Algebraic Basis for Sequenced Semantics

- Solution is at the level of the algebra.
  ⇒ Any language with sequenced semantics can be supported.

Temporal Query Processing

- To implement the sequenced semantics two new algebra operators for the adjustment of periods are needed:
  - Temporal normalization $\mathcal{N}$
  - Temporal alignment $\phi$
- Adjustment respects the lineage.
- Reduction rules from temporal RA to nontemporal RA.
  - “How to use the adjustment operators”
- Period propagation $\epsilon$ makes it possible to access the original periods.
  - Used for extended snapshot reducibility and scaling

Temporal Adjustment

- The purpose of a temporal adjustment operators is to break periods into pieces.
- Two temporal adjustment operators are required since there are two classes of operators in relational DBMSs:
  - One input tuple contributes to at most one result tuple per time point.
    ⇒ temporal normalization
    Example: Aggregation
  - One input tuple contributes to more than one result tuple per time point.
    ⇒ temporal alignment
    Example: Joins

Temporal Normalization/1

- Normalize splits each tuple in $R$ with respect to the tuples in $S$ that match on the grouping attributes $B$.

  $\mathcal{N}_B(R,S)$

  - Algebra: $\mathcal{N}_B(R,S)$
  - $R$ is split with respect to all relevant tuples in $S$. 

  **Diagram:**

  - $R$ and $S$ are sets of tuples.
  - $\mathcal{N}_B(R,S)$ indicates the normalization process.
  - $r$, $g_1$, $g_2$, $T_1$, $T_2$, $T_3$, $T_4$ are specific tuples or time periods.
Temporal Normalization/2

- Number of contracts per department: $D_{\text{TAVG}(\text{DUR}(T))}(R)$

- Employees managed by manager: $M \exists_{M.D=R.D} T R$

- One input tuple contributes to at most one result tuple per month.

Temporal Alignment/1

- **Align** splits each tuple in $R$ with respect to each tuple in the group of tuples in $S$ that satisfy $\theta$.

- Algebra: $\alpha(R)$

Temporal Alignment/2

- One input tuple contributes to more than one result tuple per month. E.g., $m_1$ contributes twice to month Feb.

Absorb

- The alignment produces all possibly required fragments (for Joins, LOJs, FOJs, etc). This can lead to temporal duplicates that must be eliminated.

- **Absorb** eliminates from $r$ temporal duplicates, i.e., tuples with a period that is contained in the period of a value-equivalent tuple.

- Algebra: $\alpha(R)$
**Answering Sequenced Queries/1**

Approach for formulating sequenced queries:

1. Formulate query without thinking about time/periods; make all operators temporal (e.g., $\vartheta^T$).

2. Add copies of periods that are needed later (in conditions, for scaling, etc).

3. Replace all references to periods with references to these copies.

4. Apply reduction rules to get nontemporal relational algebra expression.

---

**Table of Contents**

- Elements of the sequenced semantics
  - Snapshot reducibility
  - Change preservation
  - Extended snapshot reducibility
  - Scaling

- Integration of temporal support in DBMS kernel
  - Algebraic adjustment operators
  - Reduction rules

- SQL implementation

---

**Reduction Rules**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection $\sigma^T_\theta (R)$</td>
<td>$\sigma_\theta (R)$</td>
</tr>
<tr>
<td>Projection $\pi^T_\theta (R)$</td>
<td>$\pi_B \cdot (N_B (R, R))$</td>
</tr>
<tr>
<td>Aggregation $B^T \cdot \vartheta_F (R)$</td>
<td>$B \cdot \vartheta_F (N_B (R, R))$</td>
</tr>
<tr>
<td>Difference $R \cdot -^T S$</td>
<td>$N_A (R, S) \cdot - N_A (S, R)$</td>
</tr>
<tr>
<td>Union $R \cup^T S$</td>
<td>$N_A (R, S) \cup N_A (S, R)$</td>
</tr>
<tr>
<td>Intersection $R \cap^T S$</td>
<td>$N_A (R, S) \cap N_A (S, R)$</td>
</tr>
<tr>
<td>Cart. Prod. $R \times^T S$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
<tr>
<td>Inner Join $R \cdot \vartheta^T_\theta (S)$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
<tr>
<td>Left O. Join $R \cdot \vartheta^T_\theta (S)$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
<tr>
<td>Right O. Join $R \cdot \vartheta^T_\theta (S)$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
<tr>
<td>Full O. Join $R \cdot \vartheta^T_\theta (S)$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
<tr>
<td>Anti Join $R \cdot \vartheta^T_\theta (S)$</td>
<td>$A \cdot (N_A (R, S) \cdot \vartheta_T (S, R))$</td>
</tr>
</tbody>
</table>

---

**Answering Sequenced Queries/2**

Query: $D^T \cdot AVG(DUR(T)) (R)$

1. Timestamp propagation: $D^T \cdot AVG(DUR(T)) (\epsilon_U (R))$

2. Timestamp substitution: $D^T \cdot AVG(DUR(U)) (\epsilon_U (R))$

3. Temporal adjustment: $N_D (\epsilon_U (R), \epsilon_U (R))$

4. Nontemporal aggregation: $D \cdot T \cdot AVG(DUR(U)) (N_D (\epsilon_U (R), \epsilon_U (R)))$
**PostgreSQL Implementation/1**

- DBMS kernel integration of temporal adjustment.

**SQL**

![Database Management System (DBMS) kernel integration of temporal adjustment.

- Parser 60kloc
- Analyzer/Rewriter 50kloc
- Optimizer 50kloc
- Executor 40kloc

**DBMS**

- Files and Access Methods
- Buffer Manager
- Disk Manager
- Recovery Manager
- Lock Manager

**Data and Index Files**

**PostgreSQL Implementation/2**

- SQL extension that provides direct access to the adjustment operators:

  - $\epsilon_U(R)$: `SELECT Ts AS Us, Te AS Ue, * FROM R`
  - $N_B(R, S)$: `FROM (R NORMALIZE S USING(B)) AS R`
  - $\phi_{\theta}(R, S)$: `FROM (R ALIGN S ON $\theta$) AS R`
  - $\alpha(\phi(R, S) \Join \phi(R, S))$: `WHERE ...`

- Additional (technical) details will be added on the following slides

- Source Code: [http://www.ifi.uzh.ch/dbtg/research/align.html](http://www.ifi.uzh.ch/dbtg/research/align.html)

**Timestamp Propagation in SQL**

- The **extend** operator adds to the schema of a relation $R$ an attribute $U$ that is a duplicate of the period of $R$.

- Algebra: $\epsilon_U(R)$

- SQL:

  ```sql
  WITH
  R AS (SELECT *, Ts AS Us, Te AS Ue FROM R)
  ...```

**Temporal Normalization in SQL**

- The **normalize** operator splits each tuple in $R$ with respect to the tuples in $S$ that match on the grouping attributes $B$.

- Algebra: $N_B(R, S)$ or $N_\theta(R, S)$

- SQL:

  ```sql
  ... FROM (R NORMALIZE S USING(B) WITH (Rs, Re, Ss, Se)) AS Rnorm ...```

  ```sql
  ... FROM (R NORMALIZE S ON $\theta$
  WITH (Rs, Re, Ss, Se)) AS Rnorm ...```

- In SQL the timestamp attributes must be specified explicitly
Temporal Alignment in SQL

- The **align** operator splits each tuple in \( R \) with respect to each tuple in \( S \) that satisfies \( \theta \).

- Algebra: \( \phi_\theta(R, S) \)

- SQL:

  ```sql
  FROM ( R ALIGN S 
  ON \( \theta \)
  WITH (Rs,Re,Ss,Se) ) AS Radj
  ```

- In SQL the timestamp attributes must be specified explicitly

---

Absorb in SQL

- The **absorb** operator eliminates from \( r \) tuples with a period that is contained in the period of a value-equivalent tuple.

- Algebra: \( \alpha(\phi(R, S) \uplus \phi(R, S)) \)

- SQL:

  ```sql
  SELECT ... 
  FROM (... ) AS Radj, (... ) AS Sadj
  WHERE \( \theta \)
  AND (((Radj.Ts = Rs OR Radj.Ts = Ss) 
  AND (Radj.Te = Re OR Radj.Te = Se)) 
  OR Rs IS NULL OR Ss IS NULL) ;
  ```

- Rs and Re are the propagated timestamps of \( R \)
- Ss and Se are the propagated timestamps of \( S \)
- \( (T_S = R_s \lor T_S = S_s) \land (T_E = R_e \lor T_E = S_e) \lor R_s = \omega \lor S_s = \omega \)

---

Example/1

- \( R \)

  | \( n_1 = \text{Ann} \) | \( n_2 = \text{Joe} \) | \( n_3 = \text{Ann} \) |

- \( \epsilon_U(R) \)

  | \( n_1 = \text{Ann} \{1,8\} \) | \( n_2 = \text{Joe} \{2,6\} \) | \( n_3 = \text{Ann} \{8,12\} \) |

- \( N(\epsilon_U(R), \epsilon_U(R)) \)

  | \( (\text{Ann} \{1,8\}) \) | \( (\text{Ann} \{1,8\}) \) | \( (\text{Ann} \{1,8\}) \) |

  | \( (\text{Joe} \{2,6\}) \) | \( (\text{Ann} \{8,12\}) \) |

- \( r^\theta_{\text{AVG(DUR(U))}}(N(\epsilon_U(R), \epsilon_U(R))) \)

  | \( 7 \) | \( 5.5 \) | \( 7 \) | \( 4 \) |

---

Example/2

- Translate resulting algebra expression to SQL:

  ```sql
  T^\theta_{\text{AVG(DUR(U))}}(N(\epsilon_U(R), \epsilon_U(R)))
  ```

- Schema: \( R(N, Ts, Te) \)

  ```sql
  WITH x AS ( SELECT Ts Us, Te Ue, * FROM R )
  SELECT AVG(Ue - Us), Ts, Te
  FROM ( x AS x1 NORMALIZE x AS x2
  USING() 
  WITH (Ts,Te,Ts,Te) ) AS Rnorm
  GROUP BY Ts, Te;
  ```
Scaling in SQL/1

- Scaled values are functions of the original value, the old period, and the new period.
- The following function is a simple user-defined function that uniformly scales values.

```sql
CREATE OR REPLACE FUNCTION scale(x FLOAT, ts_new DATE, te_new DATE, ts_old DATE, te_old DATE)
RETURNS FLOAT AS $$
BEGIN
  RETURN x * (te_new - ts_new) / (te_old - ts_old);
END;
$$ LANGUAGE PLPGSQL;
```

Scaling in SQL/2

Procedure for the integration of scaling into the query processing workflow with adjustment operators:

1. propagate periods, $c$;
2. normalize, $N_\theta$ or align, $\phi_\theta$ (possibly use scaled values in $\theta$);
3. possibly scale values and remove propagated attributes;
4. apply the corresponding nontemporal operator, $\psi$.

Scaling in SQL/3

Task 4.10

Query: Determine amount of external funding per department.

$$\vartheta T : B @ scale \sum B(p)$$

$$\Rightarrow \vartheta T : B @ scale \sum U(p))$$

$$\Rightarrow D, T \vartheta \sum \{scale(B, T, U)) \} (N_D(\epsilon_U(p), \epsilon_U(p)))$$

WITH

P1 AS (SELECT Ts Us, Te Ue, * FROM P),
P2 AS (SELECT * FROM (P1 x NORMALIZE P1 y ON x.D=y.D WITH (Ts, Te, Ts, Te)) AS P),
P3 AS (SELECT D, N, scale(B, Ts, Te, Us, Ue) B, Ts, Te FROM P2)

SELECT D, SUM(B), Ts, Te
FROM P3
GROUP BY D, Ts, Te;

Summary

- Sequenced semantics
- Snapshot reducibility
- Change preservation
- Extended snapshot reducibility
- Scaling
- New algebraic adjustment operators in the DBMS kernel that provide support for the sequenced semantics.
  - normalize
  - align
  - extend
  - absorb
- Timestamp propagation
- Direct mapping of the adjustment operators to SQL for illustration purposes. Other SQL extensions are possible.