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

Table of Contents

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 $[T_S, T_E)$
- $t \in T \equiv T_S \leq t < T_E$

Timeslice and Snapshot Equivalence

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

$$\tau_t(r) \quad = \quad \{z^{(n)} \mid \exists x \in r(z.\mathbf{A} = x.\mathbf{A} \wedge x.T_S \leq t < x.T_E)\}$$

- Two temporal relations, r and s, are snapshot equivalent, $r \overset{s}{\equiv} s$, iff for all times t their snapshots are identical.
- Definition of **snapshot equivalence:**

$$r \overset{s}{\equiv} s \quad \text{iff} \quad \forall t(\tau_t(r) = \tau_t(s))$$

Example of Snapshot Equivalence

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

checkout1

CustID	**TapeNum**	T
C102	T1245	[19,20]
C102	T1245	[21,22]

checkout2

CustID	**TapeNum**	T
C102	T1245	[19,22]

  - Apply the timeslice operator at each time point:

$$\tau_{19}(checkout1) = \{(C102, T1245)\} \qquad \tau_{19}(checkout2) = \{(C102, T1245)\}$$
$$\tau_{20}(checkout1) = \{(C102, T1245)\} \qquad \tau_{20}(checkout2) = \{(C102, T1245)\}$$
$$\tau_{21}(checkout1) = \{(C102, T1245)\} \qquad \tau_{21}(checkout2) = \{(C102, T1245)\}$$
$$\tau_{22}(checkout1) = \{(C102, T1245)\} \qquad \tau_{22}(checkout2) = \{(C102, T1245)\}$$

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

Snapshot Reducibility

Snapshot reducibility reduces the semantics of temporal operators to the semantics of the corresponding nontemporal operators.

ψ^T is **snapshot reducible** to ψ iff for all t:

$$\tau_t(\psi^T(R_1, \ldots, R_n)) \equiv \psi(\tau_t(R_1), \ldots, \tau_t(R_n))$$

Illustration of snapshot reducibility:

- $\forall t : \tau_t(\psi^T(D^T)) = \psi(\tau_t(D^T))$

$$\begin{tikzcd} D^T \arrow[r, "\psi^T"] \arrow[d, "\tau_t"'] & R^T \arrow[d, "\tau_t"] \\ D_t \arrow[r, "\psi"] & R_t \end{tikzcd}$$

- D^T = temporal DB
- $\psi^T \in \{\sigma^T, \pi^T, \vartheta^T, \times^T, \cup^T, -^T\}$
- R^T = temporal result relation
- τ_t = snapshot at time point t
- D_t = snapshot of D^T at time t
- $\psi \in \{\sigma, \pi, \vartheta, \times, \cup, -\}$
- R_t = result relation at time t

Reducibility of Temporal Operators

- For each non-temporal operator there is a snapshot reducible operator:
  - count at each point in time
  - join at each point in time
  - primary key at each point in time
  - ranking at each point in time
- We write ψ^T for the snapshot counterpart of the relational algebra operator ψ.
- Ex: A temporal count ϑ^T corresponds to a nontemporal count ϑ at each point in time.
- **Snapshot reducibility** ensures a systematic generalization of all nontemporal operators to temporal operators.
  - ${}_D\vartheta_{SUM} \ldots \Rightarrow {}_D\vartheta^T_{SUM} \ldots$
  - $\bowtie_\theta \ldots \Rightarrow \bowtie^T_\theta \ldots$

Data Lineage/1

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

$$\mathsf{L}[\sigma_\theta^T(R)](z,t) = \{\langle r \rangle \mid r \in R \wedge z.\mathbf{A} = r.\mathbf{A} \wedge t \in r.T\}$$

$$\mathsf{L}[\pi_{\mathbf{B}}^T(R)](z,t) = \{\langle r \rangle \mid r \in R \wedge z.\mathbf{B} = r.\mathbf{B} \wedge t \in r.T\}$$

$$\mathsf{L}[{}_{\mathbf{B}}\vartheta_F^T(R)](z,t) = \{\langle r \rangle \mid r \in R \wedge z.\mathbf{B} = r.\mathbf{B} \wedge t \in r.T\}$$

$$\mathsf{L}[R -^T S](z,t) = \{\langle r, \bot \rangle \mid r \in R \wedge z.\mathbf{A} = r.\mathbf{A} \wedge t \in r.T\}$$

$$\mathsf{L}[R \cup^T S](z,t) = \{\langle r, \bot \rangle \mid r \in R \wedge z.\mathbf{A} = r.\mathbf{A} \wedge t \in r.T\} \cup \{\langle \bot, s \rangle \mid s \in S \wedge z.\mathbf{A} = s.\mathbf{A} \wedge t \in s.T\}$$

$$\mathsf{L}[R \times^T S](z,t) = \{\langle r, s \rangle \mid r \in R \wedge z.\mathbf{A} = r.\mathbf{A} \wedge t \in r.T \wedge s \in S \wedge z.\mathbf{C} = s.\mathbf{C} \wedge t \in s.T\}$$

$$\mathsf{L}[R \rhd_\theta^T S](z,t) = \{\langle r, \bot \rangle \mid r \in R \wedge z.\mathbf{A} = r.\mathbf{A} \wedge t \in r.T\}$$

Data Lineage/2

Consider query $Q = {}_{D}\vartheta^{T}_{CNT(*)}(R)$:

R

	N	D	B	T
r_1	P_1	CS	$5K$	$[1,6)$
r_2	P_2	CS	$6K$	$[4,7)$
r_3	P_3	MA	$2K$	$[1,3)$

	D	CNT	T
z_1	CS	1	$[1,4)$
z_2	CS	2	$[4,6)$
z_3	CS	1	$[6,7)$
z_4	MA	1	$[1,3)$

For z_1, z_2, and z_3 and time points 3, 4, 5 and 6, we get the following lineage sets:

- $\mathrm{L}[{}_{D}\vartheta^{T}_{CNT(*)}(R)](z_1, 3) = \{\langle r_1 \rangle\}$
- $\mathrm{L}[{}_{D}\vartheta^{T}_{CNT(*)}(R)](z_2, 4) = \{\langle r_1 \rangle, \langle r_2 \rangle\}$
- $\mathrm{L}[{}_{D}\vartheta^{T}_{CNT(*)}(R)](z_2, 5) = \{\langle r_1 \rangle, \langle r_2 \rangle\}$
- $\mathrm{L}[{}_{D}\vartheta^{T}_{CNT(*)}(R)](z_3, 6) = \{\langle r_2 \rangle\}$

Change Preservation

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

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

---

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

$$\begin{aligned}
&\forall t, t' \in z.T(\mathrm{L}(z,t) = \mathrm{L}(z,t')) \wedge \\
&(z.T_S - 1 \in z'.T \Rightarrow \mathrm{L}(z', z.T_S - 1) \neq \mathrm{L}(z, z.T_S)) \wedge \\
&(z.T_E \in z'.T \Rightarrow \mathrm{L}(z', z.T_E) \neq \mathrm{L}(z, z.T_S))
\end{aligned}$$

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

- π_D^T

Extended Snapshot Reducibility

We want to access time intervals in snapshot reducible operators.

- at each time point join long and short projects

ψ^T is **extended snapshot reducible** iff for all t:

$$\tau_t(\psi^T(R_1, \ldots, R_n)) \equiv \pi_{\mathbf{E}}(\psi(\tau_t(\epsilon_{U_1}(R_1)), \ldots, \tau_t(\epsilon_{U_n}(R_n)))).$$

For $\psi \in \{\vartheta, \sigma, \pi, \times, \bowtie, ⟕, ⟖, \triangleright, \triangleright\}$, $\mathbf{E}$ = 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 ϵ).

- at each time point the average duration of externally funded projects: ${}_D\vartheta^T_{AVG(DUR(T))}(R)$

Scaling/1

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

p

D	N	B	T_S	T_E
DB	1	$181K$	2013/2/1	2013/8/1
DB	2	$196K$	2013/5/1	2014/1/1
AI	1	$153K$	2013/4/1	2013/9/1
AI	2	$120K$	2013/4/1	2013/9/1

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

r

D	B	T_S	T_E
DB	$89K$	2013/2/1	2013/5/1
DB	$165.6K$	2013/5/1	2013/8/1
DB	$122.4K$	2013/8/1	2014/1/1
AI	$273K$	2013/4/1	2013/9/1

- ${}_{D}\vartheta^{T:B@scale}_{SUM(B)}(R)$

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

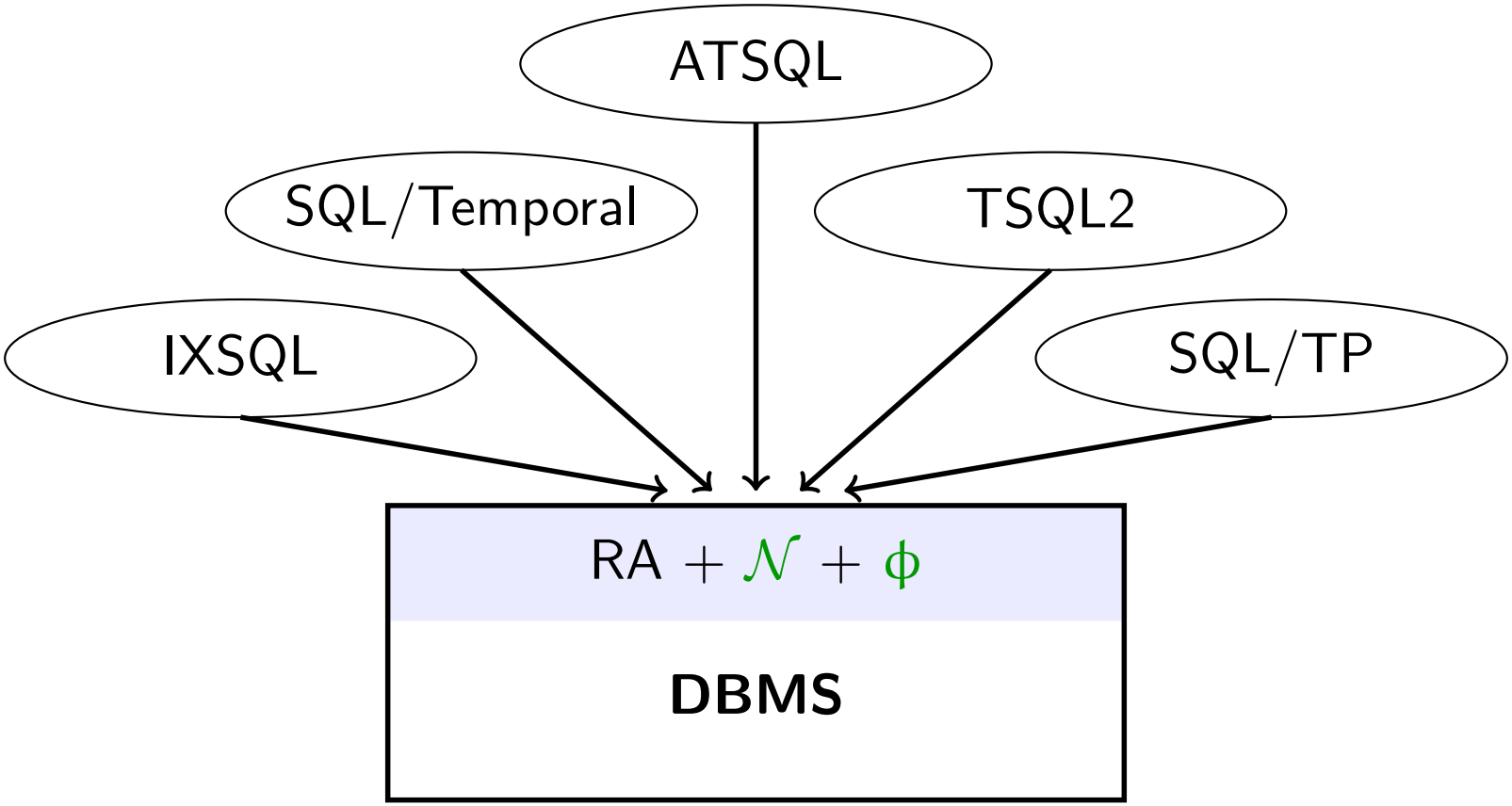
Integration of the Sequenced Semantics into the DBMS Kernel

- 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.
  $\Rightarrow$ 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 ϕ
- Adjustment respects the lineage.

- **Reduction rules** from temporal RA to nontemporal RA.
  - "How to use the adjustment operators"

- **Period propagation** ϵ makes it possible to access the original periods.
  - Used for extended snapshot reducibility and scaling

Temporal Adjustment

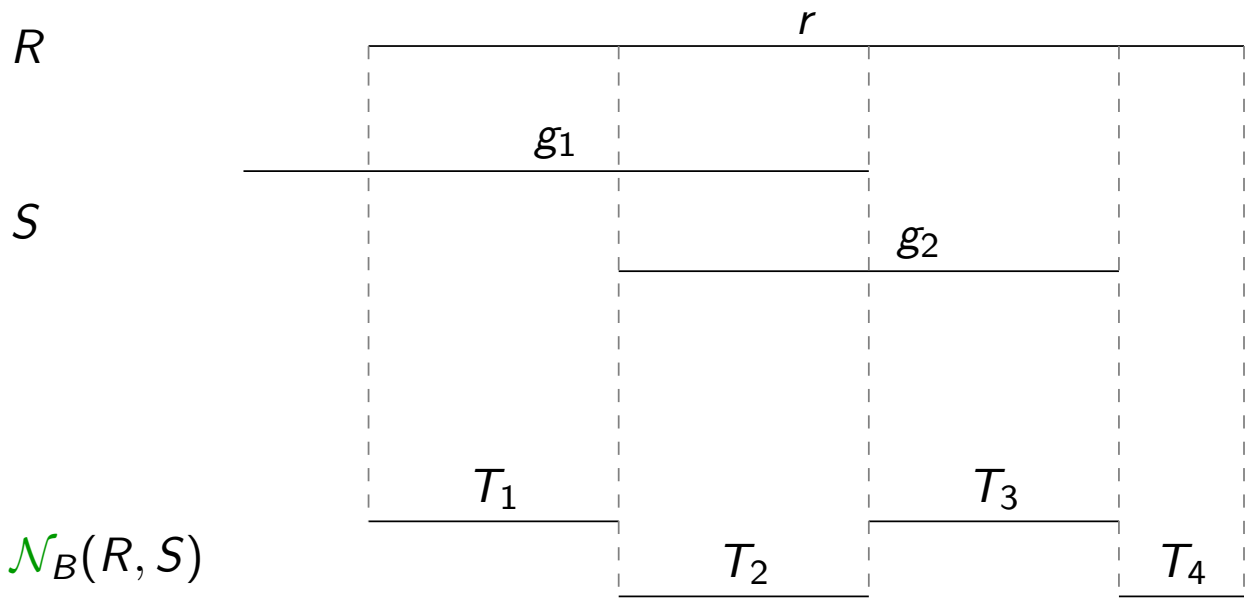
- The purpose of a temporal adjustment operators is to break periods into pieces.

- 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.
    $\Rightarrow$ **temporal normalization**
    Example: Aggregation

  - One input tuple contributes to **more than one** result tuple per time point.
    $\Rightarrow$ **temporal alignment**
    Example: Joins

Temporal Normalization/1

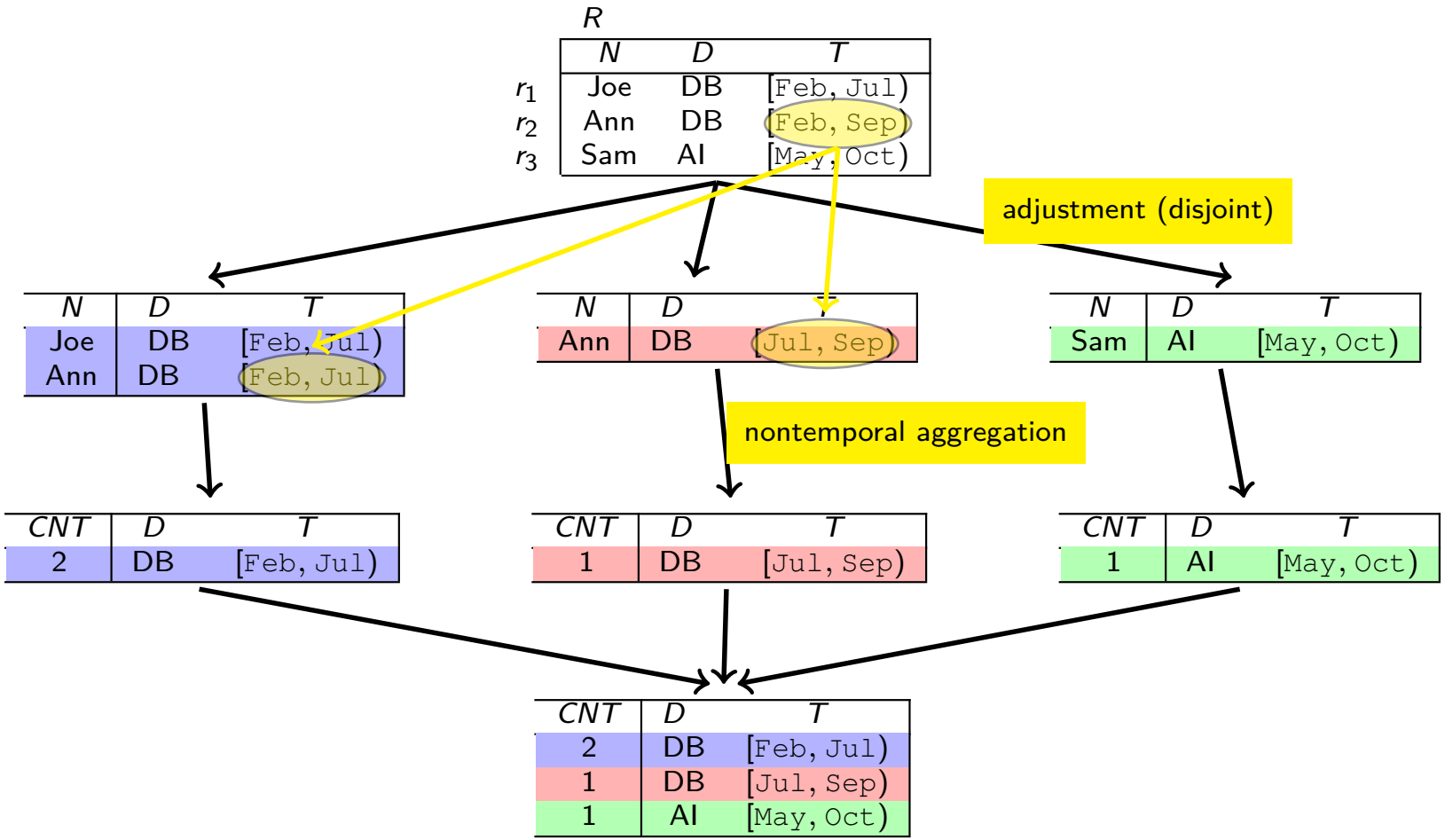
- **Normalize** splits each tuple in R with respect to the tuples in S that match on the grouping attributes B.



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

Temporal Normalization/2

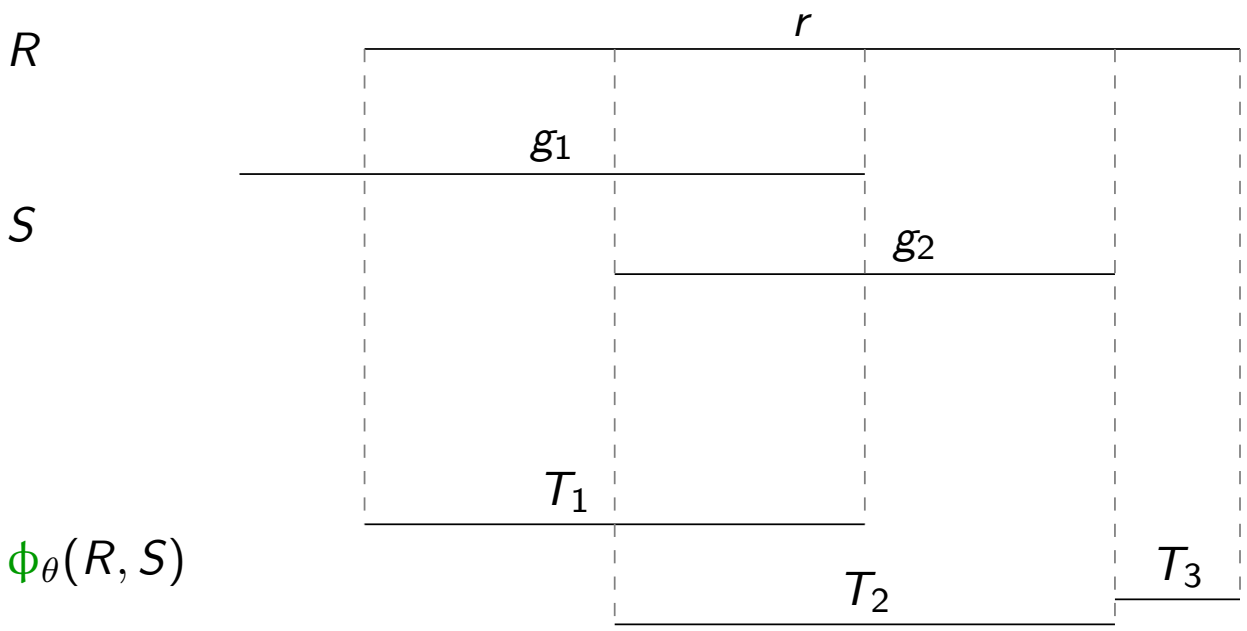
- Number of contracts per department: ${}_{D}\vartheta^{T}_{AVG(DUR(T))}(R)$



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

Temporal Alignment/1

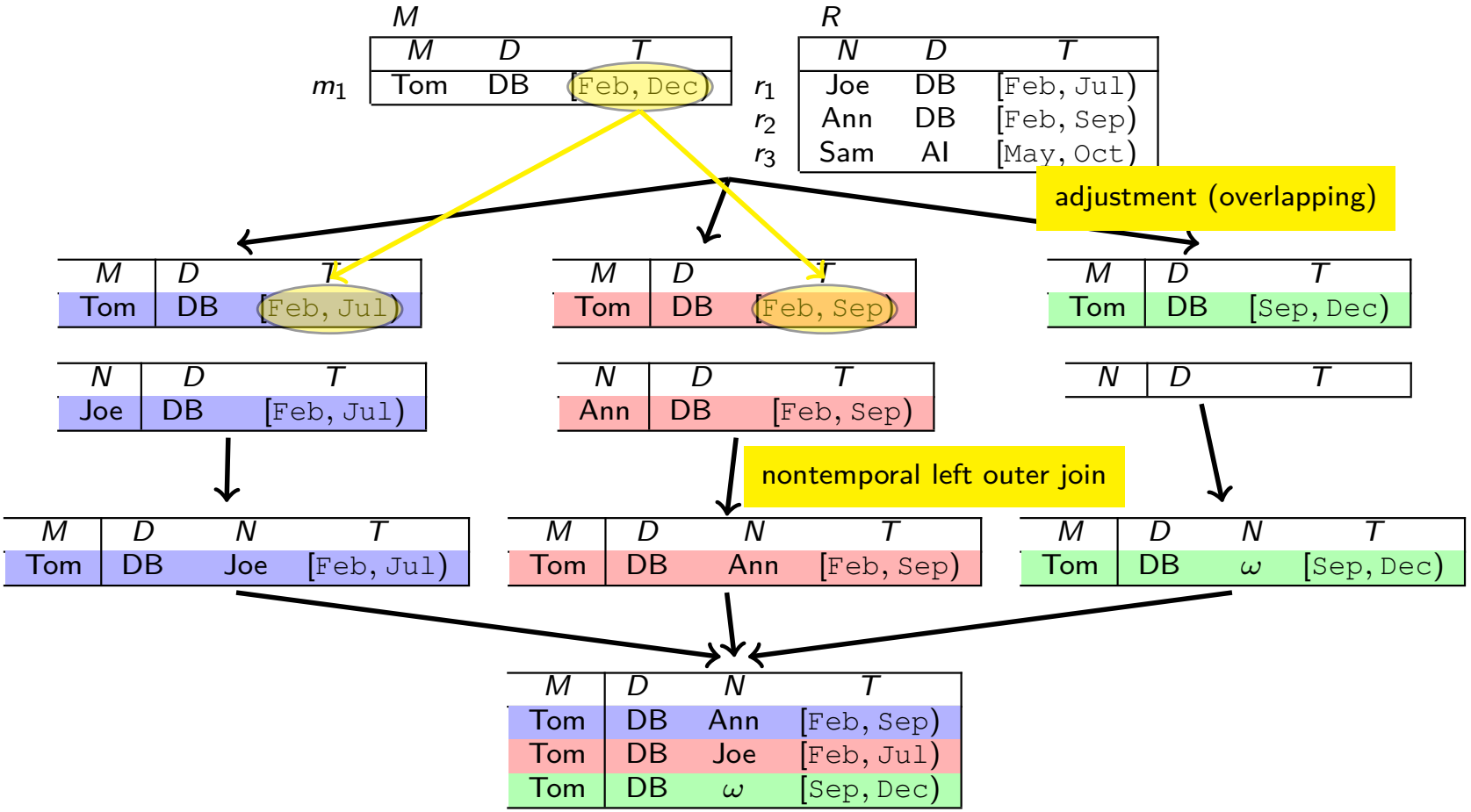
- **Align** splits each tuple in R with respect to each tuple in the group of tuples in S that satisfy θ.



- Algebra: $\phi_\theta(R, S)$
- R is split with respect to each relevant tuple in S.

Temporal Alignment/2

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



- 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.

R

A	B	T
a	c	$[1, 9)$
a	c	$[3, 7)$
a	d	$[3, 7)$
b	c	$[3, 7)$
b	d	$[3, 7)$

$\alpha(R)$

A	B	T
a	c	$[1, 9)$
a	d	$[3, 7)$
b	c	$[3, 7)$
b	d	$[3, 7)$

- Algebra: $\alpha(R)$

Reduction Rules

Operator			Reduction
Selection	$\sigma_\theta^T(R)$	$=$	$\sigma_\theta(R)$
Projection	$\pi_{\mathbf{B}}^T(R)$	$=$	$\pi_{\mathbf{B},T}(\mathcal{N}_{\mathbf{B}}(R,R))$
Aggregation	${}_{\mathbf{B}}\vartheta_F^T(R)$	$=$	${}_{\mathbf{B},T}\vartheta_F(\mathcal{N}_{\mathbf{B}}(R,R))$
Difference	$R -^T S$	$=$	$\mathcal{N}_{\mathbf{A}}(R,S) - \mathcal{N}_{\mathbf{A}}(S,R)$
Union	$R \cup^T S$	$=$	$\mathcal{N}_{\mathbf{A}}(R,S) \cup \mathcal{N}_{\mathbf{A}}(S,R)$
Intersection	$R \cap^T S$	$=$	$\mathcal{N}_{\mathbf{A}}(R,S) \cap \mathcal{N}_{\mathbf{A}}(S,R)$
Cart. Prod.	$R \times^T S$	$=$	$\alpha(\phi_\top(R,S) \bowtie_{R.T=S.T} \phi_\top(S,R))$
Inner Join	$R \bowtie_\theta^T S$	$=$	$\alpha(\phi_\theta(R,S) \bowtie_{\theta \wedge R.T=S.T} \phi_\theta(S,R))$
Left O. Join	$R ⟕_\theta^T S$	$=$	$\alpha(\phi_\theta(R,S) ⟕_{\theta \wedge R.T=S.T} \phi_\theta(S,R))$
Right O. Join	$R ⟖_\theta^T S$	$=$	$\alpha(\phi_\theta(R,S) ⟖_{\theta \wedge R.T=S.T} \phi_\theta(S,R))$
Full O. Join	$R ⟗_\theta^T S$	$=$	$\alpha(\phi_\theta(R,S) ⟗_{\theta \wedge R.T=S.T} \phi_\theta(S,R))$
Anti Join	$R \triangleright_\theta^T S$	$=$	$\phi_\theta(R,S) \triangleright_{\theta \wedge R.T=S.T} \phi_\theta(S,R)$

Answering Sequenced Queries/1

Approach for formulating sequenced queries:

1. Formulate query without thinking about time/periods; make all operators temporal (e.g., ϑ^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.

Answering Sequenced Queries/2

Query: ${}_{D}\vartheta^{T}_{AVG(DUR(T))}(R)$

1. Timestamp propagation:
   ${}_{D}\vartheta^{T}_{AVG(DUR(T))}(\epsilon_U(R))$

2. Timestamp substitution:
   ${}_{D}\vartheta^{T}_{AVG(DUR(U))}(\epsilon_U(R))$

3. Temporal adjustment:
   $\mathcal{N}_D(\epsilon_U(R), \epsilon_U(R))$

4. Nontemporal aggregation:
   ${}_{D,T}\vartheta_{AVG(DUR(U))}(\mathcal{N}_D(\epsilon_U(R), \epsilon_U(R)))$

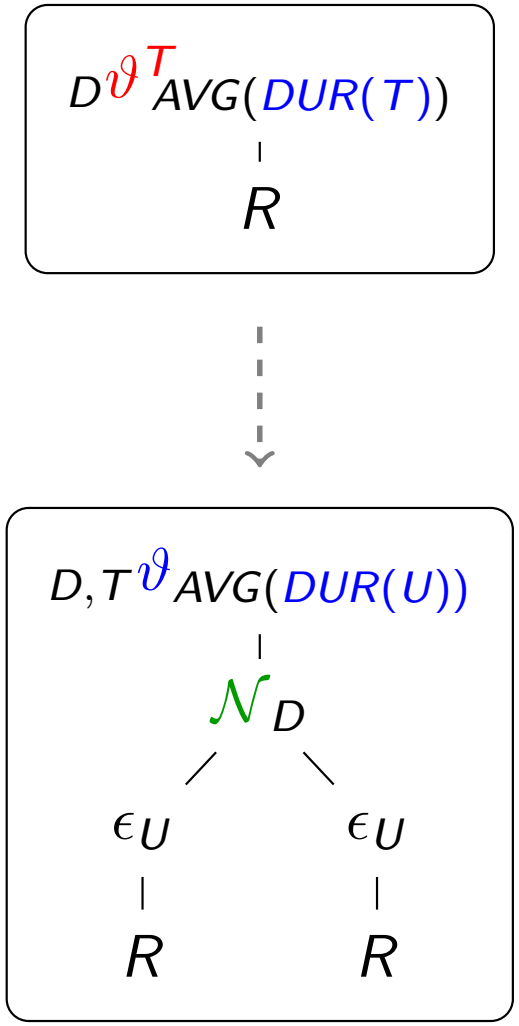
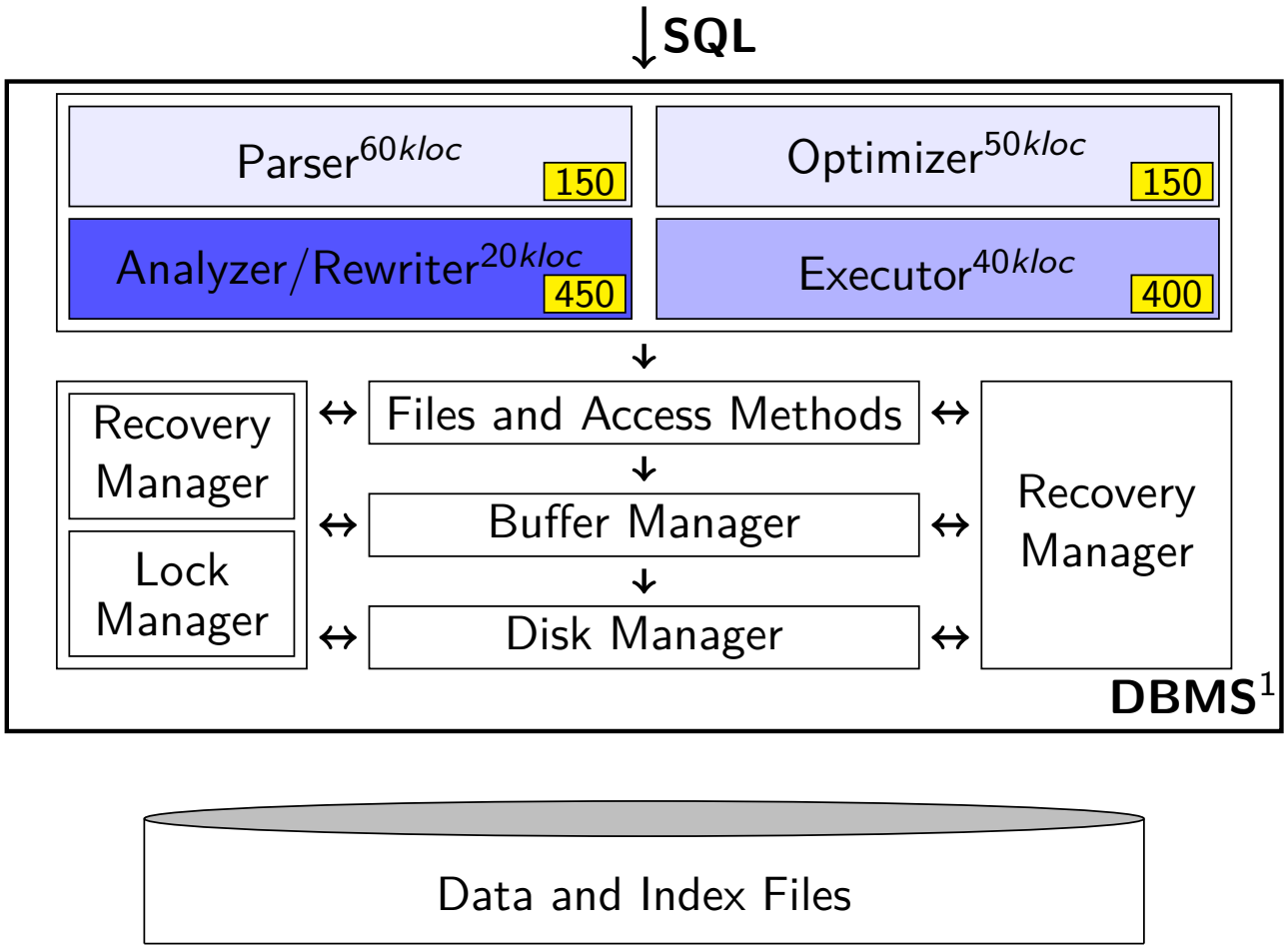


Table of Contents

PostgreSQL Implementation/1

- DBMS kernel integration of temporal adjustment.



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`

  $\mathcal{N}_{\mathbf{B}}(R,S)$: `FROM (R NORMALIZE S USING(B)) AS R`

  $\phi_\theta(R,S)$: `FROM (R ALIGN S ON` θ`) AS R`

  $\alpha(\phi(R,S) \bowtie \phi(R,S))$: `WHERE ...`

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

- Source Code:
  `http://www.ifi.uzh.ch/dbtg/research/align.html`

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:

```
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: $\mathcal{N}_B(R, S)$ or $\mathcal{N}_\theta(R, S)$
- SQL:

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

... FROM (R NORMALIZE S
          ON θ
          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 θ.
- Algebra: $\phi_\theta(R, S)$
- SQL:

```
FROM ( R ALIGN S
       ON θ
       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: $\alpha(\phi(R,S) \bowtie_\theta \phi(R,S))$
- SQL:

```
SELECT ...
FROM (...) AS Radj, (...) AS Sadj
WHERE θ
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
- $(T_S = R_s \vee T_S = S_s) \wedge (T_E = R_e \vee T_E = S_e) \vee R_s = \omega \vee S_s = \omega$

Example/1

R

$r_1 = (\text{Ann})$

$r_2 = (\text{Joe})$ $\quad$ $r_3 = (\text{Ann})$

$\epsilon_U(R)$

$r_1 = (\text{Ann}, [1, 8))$

$r_2 = (\text{Joe}, [2, 6))$ $\quad$ $r_3 = (\text{Ann}, [8, 12)$

$\mathcal{N}(\epsilon_U(R), \epsilon_U(R))$

$(\text{Ann}, [1, 8))$

$(\text{Ann}, [1, 8))$ $\quad$ $(\text{Ann}, [1, 8))$

$(\text{Joe}, [2, 6))$ $\quad$ $(\text{Ann}, [8, 12))$

${}_T\vartheta_{AVG(DUR(U))}(\mathcal{N}(\epsilon_U(R), \epsilon_U(R)))$

(7) $\quad$ (5.5) $\quad$ (7) $\quad$ (4)

Example/2

Translate resulting algebra expression to SQL:

- ${}_{T}\vartheta_{AVG(DUR(U))}(\mathcal{N}(\epsilon_U(R), \epsilon_U(R)))$
- Schema: R(N, Ts, Te)
  -
```
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.

```
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, ϵ;
2. normalize, $\mathcal{N}_\theta$ or align, Φ_θ (possibly use scaled values in θ);
3. possibly scale values and remove propagated attributes;
4. apply the corresponding nontemporal operator, ψ.

Scaling in SQL/3

Query: Determine amount of external funding per department.

$\vartheta^{T:B@scale}_{SUM(B)}(p)$

$\Rightarrow \vartheta^{T:B@scale}_{SUM(B)}(\epsilon_U(p))$

$\Rightarrow_{D,T} \vartheta_{SUM(scale(B,T,U))}(\mathcal{N}_D(\epsilon_U(p), \epsilon_U(p)))$

```
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.