Database Technology for Long Data

Michael Böhlen
Department of Computer Science

ETH Zürich	CH	1990 - 1994
University of Arizona	USA	1994 - 1995
Aalborg University	DK	1995 - 2003
Free University of Bolzano	IT	2003 - 2009
University of Zürich	CH	2009 - now

Outline

From Big Data to Long Data

Samuel Arbesman, Stop Hyping Big Data and Start Paying Attention to 'Long Data', Wired 2013.

- *Sure, big data is a powerful lens for looking at our world. Despite its limitations and requirements, crunching big numbers can help us learn a lot about ourselves.*
- *What we need to focus on is "long data" – information with a "massive historical sweep" that holds "tremendous potential for learning about ourselves".*

Long Data

According to Benjamin Bruce from Pitney Bowes, businesses can enhance their customer relationships by embracing many of Arbesman's long data ideas [Jeff Bertolucci, InformationWeek 2013]:

- *Big data is more about taking a slice in time across many different channels. But long data involves looking at information on a much longer timescale.*
- *Many companies have data that goes back 10, 20, 30 years. A longer-term view can provide information that businesses might miss if they examine data that only goes back five years or less.*
- *There's no international standards definition of big data. But the most popular description – Gartner's 3V of high volume, high velocity, and high variety – doesn't explicitly include historical information as a key component of big data.*

Characteristics of Time

Ubiquitous: All information is **qualified** with a time period

- Web, RDF stores (triples should really be quintuples)
- medical records, loans, ...
- transport information (Google Transit, Dijkstra)
- ...

Gain: Additional and more precise information

- Prediction
- Analysis
- Strategy planning
- Accountability

Example 1: Temporal Key

- A **key** is a column whose value identifies a tuple in a relation. Typical keys: account nr, SSN, phone nr, etc.
- The phone number must be unique **at each point in time**.

PhoneBook

Name	Address	PhoneNr	Start	End
Tom	ZH	+41 62 625 22 14	1978/5	2005/6
Pam	BE	+41 31 211 14 23	2000/1	2006/5
Leo	LU	+41 62 625 22 14	2006/6	now
Ann	AG	+41 31 211 14 23	2004/7	now

  - Third row is OK since phone number is reused at a later time.
  - Fourth row is not OK since phone number is reused at the same time.
- Note: SQL cannot specify such keys.
- Li, Snodgrass, Deng, Gattu, Kasthurirangan: *Efficient Sequenced Integrity Constraint Checking*, ICDE 2001.

Example 2: Temporal Query

- **Input**: Relation with external project funding

p

	N	D	B	T
r_1	Tom	CS	181K	[Feb, Aug)
r_2	Pam	CS	184K	[May, Aug)
r_3	Jud	M	153K	[Apr, Sep)

- **Query**: What is the amount of external funding per department?

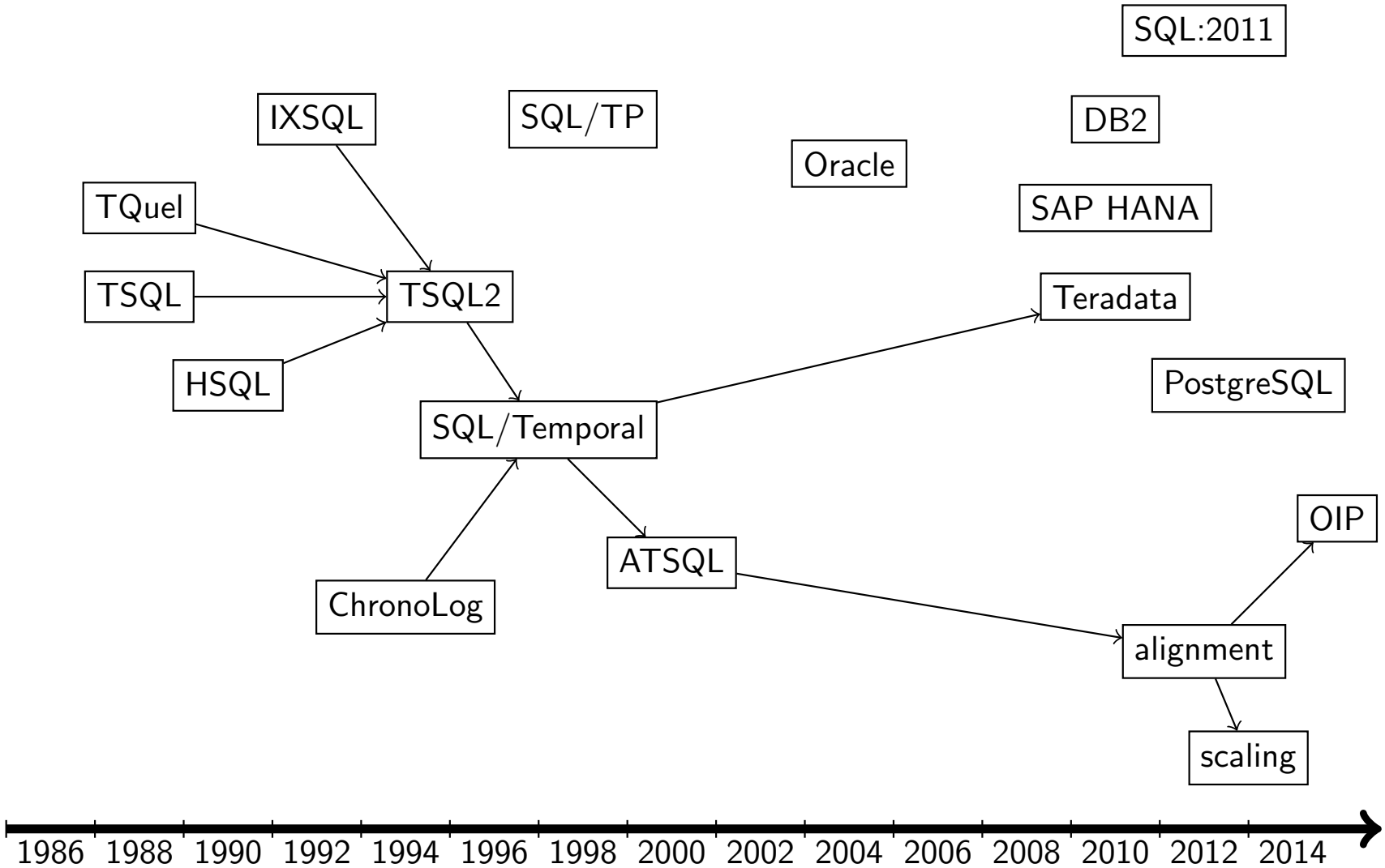
- **Output**: Temporal (at each point in time) aggregation

result

	D	B	T
z_1	CS	89K	[Feb, May)
z_2	CS	276K	[May, Aug)
z_3	M	153K	[Apr, Sep)

What database system functionality is needed to get such results?

Overview


SQL:2011
IXSQL
SQL/TP
DB2
Oracle
TQuel
SAP HANA
TSQL
TSQL2
Teradata
HSQL
PostgreSQL
SQL/Temporal
OIP
ATSQL
ChronoLog
alignment
scaling
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014

Oracle 10g, 2003

- Temporal extensions are added via workspace manager. Compliant with SQL:2011 as of Oracle 12c.
- `http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28396.pdf`
- compression, ...
- Support for **time travel** through flashback technology:
  - `WHERE AS OF TIMESTAMP`
  - `SetValidTime`

```
CREATE TABLE e (N VARCHAR(9) PRIMARY KEY, S NUMBER);
EXECUTE DBMS_WM.EnableVersioning ('e', ...);
INSERT INTO e VALUES('Ed', 30, WMSYS.WM_PERIOD(1990,2005));

EXECUTE DBMS_WM.SetValidTime(2003-01, DBMS_WM.UNTIL_CHANGED);
UPDATE e SET salary = 45000 WHERE N = 'Baxter';
```

support for temporal keys, time travel
no support for temporal queries

SAP HANA, 2010

- SAP HANA History table

p

Row	ID	Name	City	$validfrom$	$validto$
1	1001	Tina	Berlin	2013-10-01 08:30	2014-01-10 10:00
2	1002	Philip	London	2013-10-25 11:30	?
3	1003	John	New York	2013-11-05 09:00	2014-01-10 10:00
4	1004	John	Miami	2014-01-10 10:00	?

- time travel queries

```
SELECT * FROM p AS OF UTCTIMESTAMP '2014-02-02 04:11:27.73';
```

support of time travel
no support for temporal keys, temporal queries

IBM DB2 10, 2010

- Temporal extension added as of IBM DB2 10 for Z/OS
  - Support for **time travel**:
    - `SYSTEM_TIME AS OF`
    - `SYSTEM_TIME FROM...TO...`
    - `SYSTEM_TIME BETWEEN...AND...`
  - Technology: Current and history tables
  - business (= valid) time, system (= transaction) time
  - `https://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/`

support for temporal keys, time travel
no support for temporal queries

Teradata 13.10, 2010

- Temporal support added as of Teradata 13.10
  - `http://www.info.teradata.com/do_redirect.cfm?itemid=102320064`
  - Currently DB with most support for time
  - Time travel similar to IBM DB2 and Oracle DB
  - Implements parts of ANSI SQL/Temporal
  - Technology: Translation of queries at SQL level
  - Al-Kateb, Ghazal, Crolotte, Bhashyam, Chimanchode, Pakala, *Temporal query processing in Teradata*, EDBT 2013.

support of temporal keys, time travel
partial support for temporal queries

SQL:2011

- No new data type for periods; instead period definition is metadata to tables

```
CREATE TABLE Emp ( ENo INTEGER, EStart DATE, EEnd DATE,
                   EDept INTEGER,
                   PERIOD FOR EPeriod (EStart, EEnd) )
```

- system time, application time
- temporal primary and foreign key

```
ALTER TABLE Emp ADD PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS)
```

- Queries are formulated with predicates:

```
CONTAINS, OVERLAPS, SUCCEEDS, IMMEDIATELY PRECEDES, ...
```

support for temporal keys, time travel
no support for temporal queries

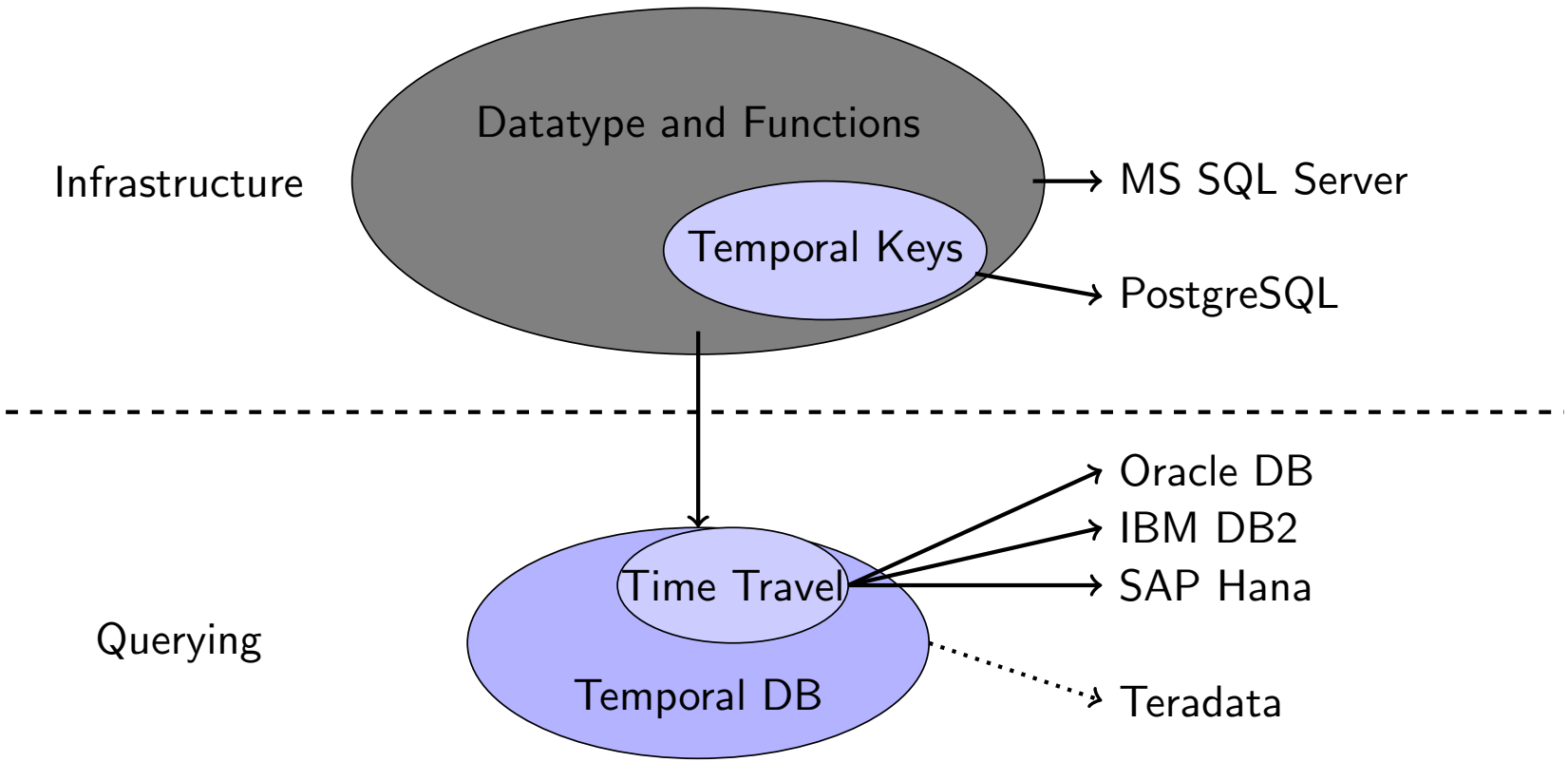
PostgreSQL 9.2, 2012

- Jeff Davis - Temporal Postgres (2007)
  - `http://temporal.projects.pgfoundry.org/`
  - Period datatype and UDF functions on periods
  - Indexing via GiST index
- PostgreSQL release 9.2 (2012)
  - `http://www.postgresql.org/docs/9.2/static/rangetypes.html`
  - Range Types
  - Indexing via GiST or SP-GiST index
  - Constraints on Ranges, i.e., temporal key constraints

support for temporal keys
no support for time travel, temporal queries

Level of Support for Time

- During recent years DBMS companies have added a lot in terms of infrastructure.
- One of the main new features of SQL:2011 is the support for time.



Take Home Messages

- During the past couple of years commercial database management systems have substantially progressed their **infrastructure** to support temporal information.
- **Time periods** (or time ranges) are supported by most systems.
- Temporal **primary keys** and temporal **foreign keys** are supported by some systems.
- Many database systems support **time travels** (going back and forth in time)
- Temporal querying by and large is **not supported**
  - Temporal predicates (CONTAIN, BEFORE, ...) and functions (DURATION, ...) are limited and not sufficient.
  - Aggregation, scaling, etc is not supported

Outline

1. History and Context

2. **Application Requirements, Goal**
   - Three Properties
   - Goal

3. Temporal Primitives

4. Mapping to SQL

5. Summary and Outlook

Temporal Data Example

- **Input**: Relation with external project funding

p

	N	*B*	*D*	U	*T*
r_{1a}	Tom	181K	CS	[Feb, Aug)	[Feb, May)
r_{1b}	Tom	181K	CS	[Feb, Aug)	[May, Aug)
r_2	Pam	184K	CS	[May, Aug)	[May, Aug)
r_3	Jud	153K	M	[Apr, Sep)	[Apr, Sep)

- **Query**: What is the amount of external funding per department?
- **Result**: Temporal Aggregation ${}_{D}\vartheta^{T}_{SUM(scale(B))}(\mathbf{p})$

	D	*SUM*	*T*
z_1	CS	89K	[Feb, May)
z_2	CS	276K	[May, Aug)
z_3	M	153K	[Apr, Sep)

Timestamps must be **adjusted** for the result.

Some values must be **scaled** based on original and adjusted timestamps.

Property P1

Property P1

A period can be decomposed into a set of points (or subperiods).

Query: What are the top-2 time periods with most projects?

p

N	D	B	T_S	T_E
P1	M	10k	Jan	Dec
P2	CS	7k	Feb	Aug
P3	CS	5k	Jun	Dec

result

Cnt	T_S	T_E
3	Jun	Aug
2	Feb	Jun

Counting procedure:

... **p** @Jan

N	D	B
P1	M	10k

p @Feb

N	D	B
P1	M	10k
P2	CS	7k

p @Mar

N	D	B
P1	M	10k
P2	CS	7k

...

Property P2

Property P2

An period carries more information than a set of points.

- The following relations are semantically different:

Reagan	1981/1/20	1985/1/20
Reagan	1985/1/20	1989/1/20

$\neq$

Reagan	1981/1/20	1989/1/20

  - The first relation records the terms Reagan was elected for.
  - The second relation records the period during which Reagan was president.

- Decomposition and coalescing should be done conservatively (only when requested by the application)
- There must be a possibility to access the original timestamps

Property P3

Property P3

Sometimes information must change if the associated period changes.

Query: What is the available project funding from 2011 to 2012?

ProjX	300K	2011	2014	$\Rightarrow$	ProjX	150K	2011	2012

- We need a mechanism that allows applications to change information if the associated period is changed.
- The change is a function of
  - the original value,
  - the original time period, and
  - the new time period.

Goal

- We want a database system that provides **P1 + P2 + P3** (= sequenced semantics)
  - P1 (period can be decomposed into points)
  - P2 (period carries more information than points)
  - P3 (sometimes values must change along with associated periods)

- More specifically, we want an **algebraic solution** that supports the **sequenced semantics**.

- It is easy to get some of these properties but difficult to get all of them together.
  - SQL gives property P2
  - Research has focused on property P1
  - SQL:2011 gives property P2 and property P1 for primary and foreign keys

Take Home Messages

1. Why is SQL not good enough?
   - With SQL it is too difficult to get property P1 ($\approx$ evaluate a statement at each point in time).
2. What happens if we only support property P1 and not the other two properties?
   - Solution is not general and becomes irrelevant.
   - Can be OK to get a paper.
   - Work will have very limited impact.
3. Carefully pick a representative example.
   - aggregations is good
   - primary and foreign key are good
   - scaling is good

Outline

1. History and Context
2. Application Requirements, Goal
3. **Temporal Primitives**
   - Temporal Splitter
   - Temporal Aligner
4. Mapping to SQL
5. Summary and Outlook

Key Question

What core functionality should a database system offer to support the management of data qualified with a time period?

or: what would be my one wish from

Remarks about Database System Functionality

- We do not need a new model (**the** model) for temporal data that incorporates application semantics.
- We need **new primitives** that can be used if needed, do not impose any constraints, and support temporal querying.
- Scalability to **all** language constructs is essential.

- Applications may need
  - a primary key that holds at each point in time
  - a primary key that holds independent of time
- Applications can decide to
  - sum external funding at each point in time
  - sum external funding independent of time
- Applications can decide to
  - do XXX at each point in time
  - do XXX independent of time

The Limitation of Relational Algebra and SQL

- SQL treats periods as atomic units. This causes problems.

- What we get from a database system:

  $(\text{CS}, [\texttt{May}, \texttt{Aug})) \stackrel{?}{=} (\text{CS}, [\texttt{May}, \texttt{Aug})) \rightarrow \mathit{true}$ OK

  $(\text{CS}, [\texttt{Feb}, \texttt{May})) \stackrel{?}{=} (\text{CS}, [\texttt{May}, \texttt{Aug})) \rightarrow \mathit{false}$ OK

  $(\text{CS}, [\texttt{Feb}, \texttt{Aug})) \stackrel{?}{=} (\text{CS}, [\texttt{May}, \texttt{Aug})) \rightarrow \mathit{false}$ NOT OK

- What we want from a database system: adjusted periods that can be treated as **atomic** units and for which equality works again as expected.

Solution

- We need **two new algebra operators** (primitives) for the **adjustment** of timestamps:
  - Temporal Splitter $\mathcal{N}$
  - Temporal Aligner Φ
- We need a mechanism to **propagate timestamps** so we can access the original timestamps.

Properties of the solution:

- Systematic **reduction rules** from temporal RA to nontemporal RA with adjusted periods.
- Adjustment allows to propagate a copy of the original timestamps.
  - $\epsilon_U(\mathbf{p})$: adds a copy of the timestamp as new attribute U
- Flexible **user-defined functions** for scaling.
  - $\pi_{scale(B)}(\mathbf{p})$: scales attribute value B

Temporal Primitives

- The temporal primitives break timestamps into pieces that are aligned for the purpose of the subsequent operation.

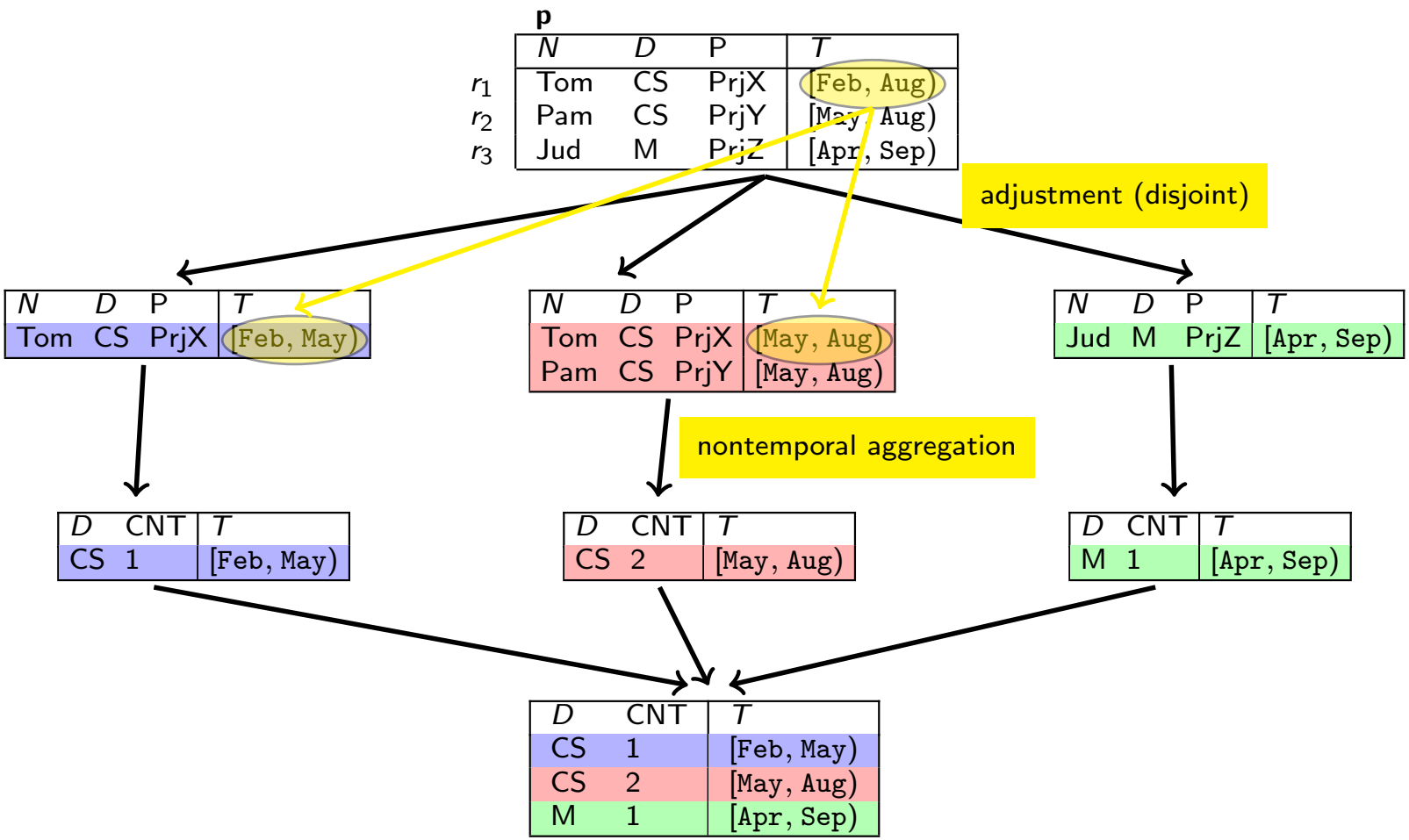
- Two temporal primitives are required:
  - One input tuple contributes to at most one result tuple at each point in time.
    $\Rightarrow$ **Temporal Splitter**
    Example: Aggregation

  - One input tuple contributes to more than one result tuple at each point in time.
    $\Rightarrow$ **Temporal Aligner**
    Example: Joins

- After a temporal primitive we can use **equality on timestamps** to get property P1.

Temporal Splitter

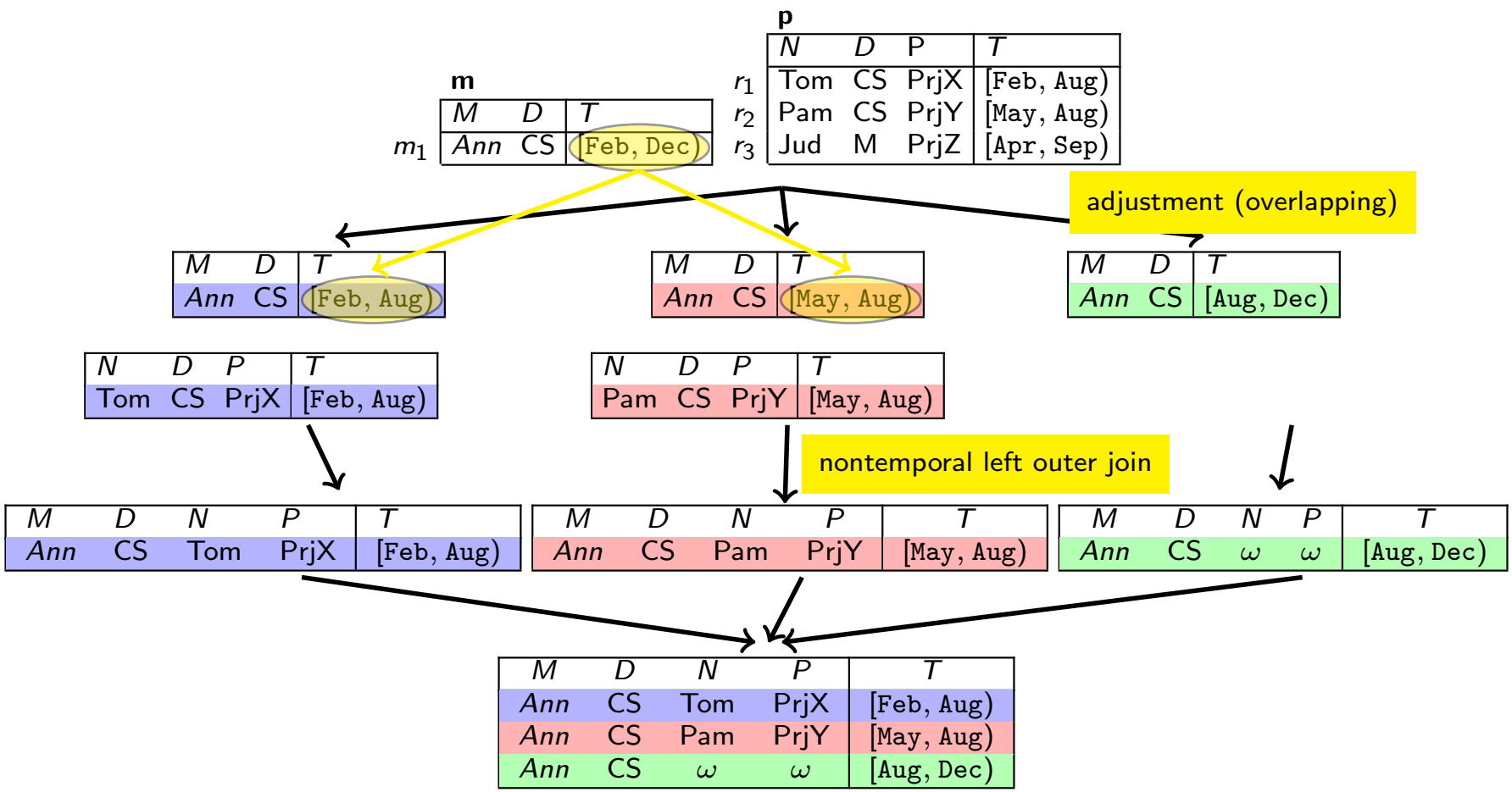
- Number of projects per department: ${}_{D}\vartheta^{T}_{COUNT(P)}(\mathbf{p})$



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

Temporal Aligner

- Managers project budgets: $\mathbf{m} \overset{T}{\bowtie}_{\mathbf{m}.D=\mathbf{p}.D} \mathbf{p}$



	M	D	T
m_1	Ann	CS	[Feb, Dec)

	N	D	P	T
r_1	Tom	CS	PrjX	[Feb, Aug)
r_2	Pam	CS	PrjY	[May, Aug)
r_3	Jud	M	PrjZ	[Apr, Sep)

M	D	T
Ann	CS	[Feb, Aug)

M	D	T
Ann	CS	[May, Aug)

M	D	T
Ann	CS	[Aug, Dec)

N	D	P	T
Tom	CS	PrjX	[Feb, Aug)

N	D	P	T
Pam	CS	PrjY	[May, Aug)

M	D	N	P	T
Ann	CS	Tom	PrjX	[Feb, Aug)

M	D	N	P	T
Ann	CS	Pam	PrjY	[May, Aug)

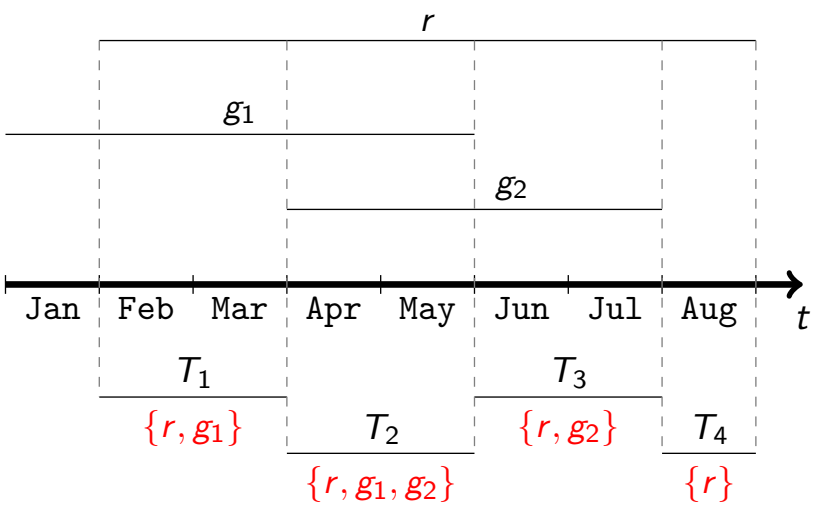
M	D	N	P	T
Ann	CS	ω	ω	[Aug, Dec)

M	D	N	P	T
Ann	CS	Tom	PrjX	[Feb, Aug)
Ann	CS	Pam	PrjY	[May, Aug)
Ann	CS	ω	ω	[Aug, Dec)

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

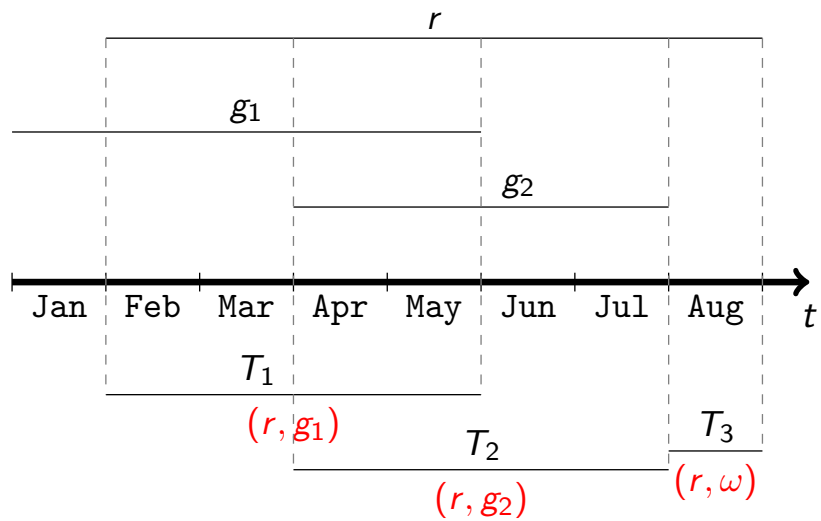
Temporal Splitter versus Temporal Aligner

Temporal Splitter



- Tuples are broken into **disjoint** pieces.
- **Groups** of matching tuples define change points.

Temporal Aligner



- Tuples are broken into **overlapping** pieces.
- **Pairs** of matching tuples define change points.

Reduction Rules: $\psi^T \longrightarrow \{\mathcal{N}, \phi\} + \psi$

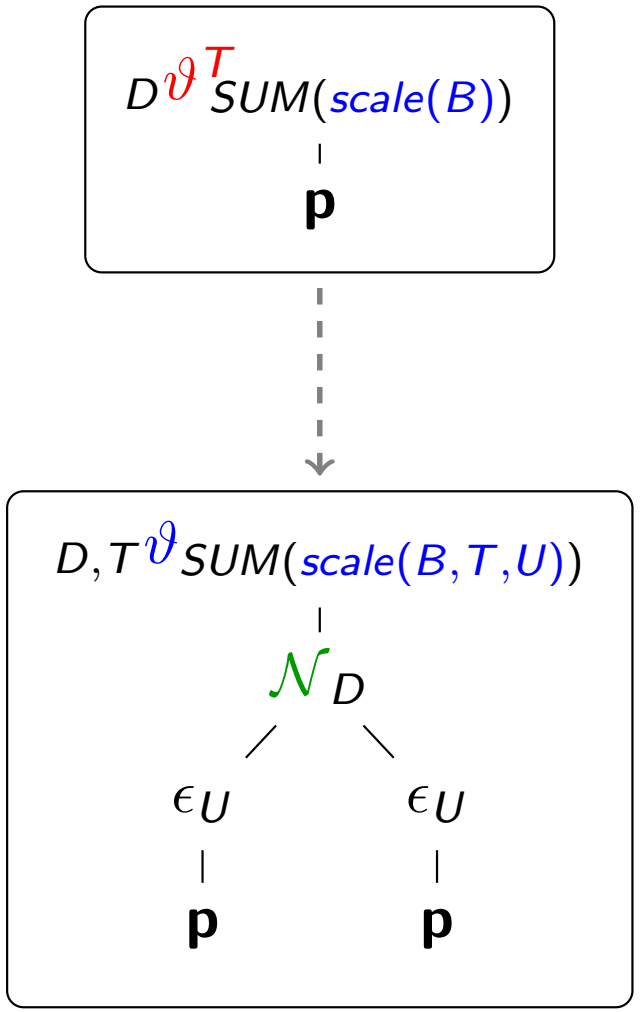
Operator			Reduction
Selection	$\sigma_\theta^T(\mathbf{r})$	$=$	$\sigma_\theta(\mathbf{r})$
Projection	$\pi_{\mathbf{B}}^T(\mathbf{r})$	$=$	$\pi_{\mathbf{B},T}(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$
Aggregation	${}_{\mathbf{B}}\vartheta_F^T(\mathbf{r})$	$=$	${}_{\mathbf{B},T}\vartheta_F(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$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

Temporal Op. $=$ Primitive + Traditional Op.

Constructing Sequenced Algebra Expressions

Query: ${}_{D}\vartheta^{T}_{SUM(scale(B))}(\mathbf{p})$

1. Timestamp propagation:
   ${}_{D}\vartheta^{T}_{SUM(scale(B))}(\epsilon_U(\mathbf{p}))$
2. Timestamp substitution:
   ${}_{D}\vartheta^{T}_{SUM(scale(B,T,U))}(\epsilon_U(\mathbf{p}))$
3. Temporal adjustment:
   $\mathbf{p}' \leftarrow \mathcal{N}_D(\epsilon_U(\mathbf{p}), \epsilon_U(\mathbf{p}))$
4. Nontemporal aggregation:
   ${}_{D,T}\vartheta_{SUM(scale(B,T,U))}(\mathbf{p}')$



Take Home Messages

1. Temporal primitives provide a principled approach to get rid of complex period comparison and use equality on adjusted periods.
2. We need two algebraic primitives for the **adjustment** of periods.
3. The algebraic primitives allow to **propagate** a copy of the original timestamps.
4. Values can be **scaled** based on original and new timestamp.
5. Physical optimizations of primitives are possible:
   - logical versus physical algebra
   - $\bowtie$ versus $\bowtie^{NL}$, $\bowtie^{HJ}$, $\bowtie^{SM}$

1. History and Context

2. Application Requirements, Goal

3. Temporal Primitives

4. **Mapping to SQL**
   - Mapping to SQL
   - Example Queries
   - Scaling Functions

5. Summary and Outlook

Mapping to SQL

- Our implementation provides direct access to primitive operators:

$$\epsilon_U(\mathbf{r}) : \texttt{SELECT Ts Us, Te Ue, * FROM r}$$
$$\mathcal{N}_\mathbf{B}(\mathbf{r},\mathbf{s}) : \texttt{FROM (r NORMALIZE s USING(B)) r}$$
$$\phi_\theta(\mathbf{r},\mathbf{s}) : \texttt{FROM (r ALIGN s ON } \theta\texttt{) r}$$
$$\alpha(\mathbf{r}) : \texttt{SELECT ABSORB * FROM r}$$

- The source code of PostgreSQL with the complete temporal functionality integrated into the kernel of PostgreSQL can be downloaded from
`http://www.ifi.uzh.ch/dbtg/research/align.html`

- Disclaimer: I am not proposing (yet another) temporal extension of SQL. The purpose is to demo a system as a proof of concept.

Query Q1

$$_{\mathbf{B}}\vartheta^{T}_{F}(\mathbf{r}) \;=\; _{\mathbf{B},T}\vartheta_{F}(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$$

Query: What is the largest X?

Input:

r

X
5
7

Answer: $\vartheta_{MAX(X)}(\mathbf{r})$

```
SELECT MAX(X)
FROM r ;
```

X
7

Input:

r

X	T
5	[Feb, Aug)
7	[May, Aug)

$$\vartheta^{T}_{MAX(X)}(\mathbf{r}) = {}_{T}\vartheta_{MAX(X)}(\mathcal{N}(\mathbf{r},\mathbf{r}))$$

```
SELECT MAX(X), Ts, Te
FROM (r NORMALIZE r USING()) r
GROUP BY Ts, Te;
```

X	T
5	[Feb, May)
7	[May, Aug)

Query Q2

$$_{\mathbf{B}}\vartheta^{T}_{F}(\mathbf{r}) \;=\; _{\mathbf{B},T}\vartheta_{F}(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$$

Query: What is the largest X per Y?

Input:

r

X	Y
5	A
7	A
3	B

Answer: $_{Y}\vartheta_{MAX(X)}(\mathbf{r})$

```
SELECT MAX(X), Y
FROM r
GROUP BY Y;
```

X	Y
7	A
3	B

Input:

r

X	Y	T
5	A	[Feb, Aug)
7	A	[May, Aug)
3	B	[Apr, Sep)

$$_{Y}\vartheta^{T}_{MAX(X)}(\mathbf{r}) = {}_{Y,T}\vartheta_{MAX(X)}(\mathcal{N}_{Y}(\mathbf{r},\mathbf{r}))$$

```
SELECT MAX(X), Y, Ts, Te
FROM (r NORMALIZE r USING(Y)) r
GROUP BY Y, Ts, Te;
```

X	Y	T
5	A	[Feb, May)
7	A	[May, Aug)
3	B	[Apr, Sep)

Query Q3

$$\mathbf{r} \triangleright^{T}_{\theta} \mathbf{s} \;=\; \Phi_{\theta}(\mathbf{r}, \mathbf{s}) \triangleright_{\theta \wedge \mathbf{r}.T = \mathbf{s}.T} \Phi_{\theta}(\mathbf{s}, \mathbf{r})$$

Query: What is the tuple with largest X?

Input:

r

X	Y
5	A
7	A
3	B

Answer: $\mathbf{r} \triangleright_{s.X > r.X} \mathbf{s}$

```
SELECT *
FROM r
WHERE NOT EXISTS
 (SELECT *
  FROM r s
  WHERE s.X > r.X);
```

X	Y
7	A

Input:

r

X	Y	T
5	A	[Feb, Aug)
7	A	[May, Aug)
3	B	[Apr, Sep)

$$\mathbf{r} \triangleright^{T}_{s.X > r.X} \mathbf{s} = \Phi_{\ldots}(\mathbf{r}, \mathbf{s}) \;\triangleright_{\ldots \wedge \mathbf{r}.T = \mathbf{s}.T}\; \Phi_{\ldots}(\mathbf{s}, \mathbf{r})$$

```
SELECT *
FROM (r ALIGN r s ON s.X > r.X) r
WHERE NOT EXISTS
 (SELECT *
  FROM (r s ALIGN r ON s.X > r.X) s
  WHERE s.X > r.X
   AND r.Ts=s.Ts AND r.Te=s.Te);
```

X	Y	T
5	A	[Feb, May)
7	A	[May, Aug)
3	B	[Aug, Sep)

Query Q4

$$_{\mathbf{B}}\vartheta^{T}_{F}(\mathbf{r}) \;=\; _{\mathbf{B},T}\vartheta_{F}(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$$

Query: What is the project budget per department?

Input:

p

N	*B*	*D*
Tom	181K	CS
Pam	184K	CS
Jud	153K	M

Answer: $_{D}\vartheta_{SUM(B)}(\mathbf{r})$

```
SELECT D, SUM(B)
FROM p
GROUP BY D;
```

D	*SUM*
CS	365K
M	153K

Input:

p

N	*B*	*D*	*T*
Tom	181K	CS	[Feb, Aug)
Pam	184K	CS	[May, Aug)
Jud	153K	M	[Apr, Sep)

$_{D}\vartheta^{T}_{SUM(scale(B))}(\mathbf{p})$

```
WITH p2 AS (SELECT Ts Us, Te Ue, * FROM p)
SELECT D, SUM(scale(B,Ts,Te,Us,Ue)), Ts, Te
FROM (p2 NORMALIZE p2 USING(D)) p3
GROUP BY D, Ts, Te;
```

D	*SUM*	*T*
CS	89K	[Feb, May)
CS	276K	[May, Aug)
M	153K	[Apr, Sep)

Scaling Function

Input:

- Value to scale: `x`
- New timestamp: [`ts_new`, `te_new`)
- Old timestamp: [`ts_old`, `te_old`)

Output:

- Scaled value of `x`

Scaling function for uniform distribution in PL/pgSQL:

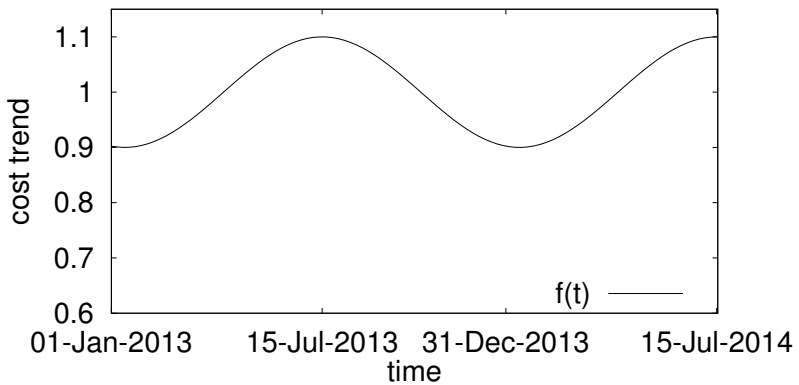
```
CREATE OR REPLACE FUNCTION
scaleU(x FLOAT, ts_new DATE, te_new DATE, ts_old DATE, te_old DATE)
RETURNS FLOAT AS $$
BEGIN
  RETURN x * (te_new - ts_new) / (te_old - ts_old);
END; $$ LANGUAGE PLPGSQL;
```

Scaling Function for Seasonal Scaling

Trend Scaling: Scaling using a function $f(t)$

$$scaleT(x, T_{new}, T_{old}) = x \cdot \frac{\int_{T_{S_{new}}}^{T_{E_{new}}} f(t) \cdot dt}{\int_{T_{S_{old}}}^{T_{E_{old}}} f(t) \cdot dt}.$$

Example: Scaling by cost of power consumption fluctuating by 20% due to cooling.



Comparison of Scaling Functions

Input:

p

N	*B*	*D*	*T*
Tom	181K	CS	[Feb, Aug)
Pam	184K	CS	[May, Aug)
Jud	153K	M	[Apr, Sep)

Query:

```
WITH p2 AS (SELECT Ts Us, Te Ue, * FROM p)
SELECT D, SUM(scale(B, Ts, Te, Us, Ue)), Ts, Te
FROM (p2 NORMALIZE p2 USING(D)) p3
GROUP BY D, Ts, Te;
```

uniform

D	*SUM*	*T*
CS	89K	[Feb, May)
CS	276K	[May, Aug)
M	153K	[Apr, Sep)

trend

D	*SUM*	*T*
CS	83.62K	[Feb, May)
CS	281.38K	[May, Aug)
M	153K	[Apr, Sep)

Take Home Messages

1. Building a publicly available system makes a huge difference.
2. The many nitty-gritties you have to sort out when building a system provide deep insights.
3. Building a system is a good investment.

4. Build a general solution that can be used by applications; do not solve a specific problem.

References

- A. Dignös, M. H. Böhlen, J. Gamper, *Overlap Interval Partition Join*, SIGMOD 2014.
- A. Dignös, M. H. Böhlen, J. Gamper, *Query Time Scaling of Attribute Values in Interval Timestamped Databases*, 29th International Conference on Data Engineering, ICDE demo 2013.
- A. Dignös, M. H. Böhlen, J. Gamper, *Temporal Alignment*, 2012 ACM SIGMOD/PODS Conference, SIGMOD 2012.
- M. H. Böhlen, J. Gamper, and C. S. Jensen. *An algebraic framework for temporal attribute characteristics*. Ann. Math. Artif. Intell., vol 46. 2006
- M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass, *Temporal Statement Modifiers*, TODS 2000.
- W. Li, R. T. Snodgrass, S. Deng, V. K. Gattu, and A. Kasthurirangan: *Efficient Sequenced Integrity Constraint Checking*, ICDE 2001.
- D. Toman, *Point-Based Temporal Extensions of SQL and Their Efficient Implementation*, Temporal Databases: Research and Practice Springer Verlag. 1998.
- N. A. Lorentzos and Y. G. Mitsopoulos, *SQL Extension for Interval Data*, TKDE 1997.

Acknowledgments

- Anton Dignös, University of Zürich
- Johann Gamper, Free University of Bozen-Bolzano
- Christian S. Jensen, Aalborg University
- Rick Snodgrass, University of Arizona
- Robert Marti, ETH and Swiss Re

- Swiss National Science Foundation
- EU (Chorochronos)

Summary

- **Goal**: sequenced semantics with P1 + P2 + P3
- **Two** primitives are required to add support for sequenced semantics to a DBMS.
  - Temporal Splitter $\mathcal{N}$
  - Temporal Aligner ϕ
- Systematic **reduction rules** from temporal RA to nontemporal RA.
- **Timestamp propagation** for accessing original timestamps.
- **User-defined functions** for scaling.
- Fully integrated into DBMS kernel of PostgreSQL.

Outlook

- **Time Series**
  - There is lots of time series data out there (from sensors)
  - Storage and compression of time series data
  - Dealing with incorrect and missing values
  - Similarity search in time series
  - Seamless integration of change (time point when event happens) and state information (time period during which a fact holds)
- **Google Knowledge Graph**
  - Xin Luna Dong at Google
  - RDF graph, (Subject,Predicate,Object) triples
  - extend triples to quintuples
  - probabilities to model reliability of information
  - time to capture age and validity of information
  - entity resolution

Outlook

- **Physical Optimization**
  - At the logical level two primitives are required to adjust periods.
  - The two primitives produce all fragments needed by regular algebra operators
  - Additional primitives can be used to boost performance
    - speed up align to not produce fragments needed for full outer join
    - add a new primitive for full outer joins.
- **Temporal Relationships**
  - Temporal primitives focus on information that is valid at the same time
  - Because of periods we can no longer easily process information that is valid at the same time; the temporal primitives give this possibility back to us.
  - Querying across multiple time (e.g., temporal patters) is interesting

Thank You!