Database Technology for Long Data

Michael Böhlen
Department of Computer Science

<table>
<thead>
<tr>
<th>Institution</th>
<th>Country</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH Zürich</td>
<td>CH</td>
<td>1990 - 1994</td>
</tr>
<tr>
<td>University of Arizona</td>
<td>USA</td>
<td>1994 - 1995</td>
</tr>
<tr>
<td>Aalborg University</td>
<td>DK</td>
<td>1995 - 2003</td>
</tr>
<tr>
<td>Free University of Bolzano</td>
<td>IT</td>
<td>2003 - 2009</td>
</tr>
<tr>
<td>University of Zürich</td>
<td>CH</td>
<td>2009 - now</td>
</tr>
</tbody>
</table>
Outline

1. History and Context
2. Application Requirements, Goal
3. Temporal Primitives
4. Mapping to SQL
5. Summary and Outlook
From Big Data to Long Data


- **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.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>PhoneNr</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>ZH</td>
<td>+41 62 625 22 14</td>
<td>1978/5</td>
<td>2005/6</td>
</tr>
<tr>
<td>Pam</td>
<td>BE</td>
<td>+41 31 211 14 23</td>
<td>2000/1</td>
<td>2006/5</td>
</tr>
<tr>
<td>Leo</td>
<td>LU</td>
<td>+41 62 625 22 14</td>
<td>2006/6</td>
<td>now</td>
</tr>
<tr>
<td>Ann</td>
<td>AG</td>
<td>+41 31 211 14 23</td>
<td>2004/7</td>
<td>now</td>
</tr>
</tbody>
</table>
```

- 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.

Example 2: Temporal Query

▶ Input: Relation with external project funding

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>D</th>
<th>B</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>Tom</td>
<td>CS</td>
<td>181K</td>
<td>[Feb, Aug)</td>
</tr>
<tr>
<td>r2</td>
<td>Pam</td>
<td>CS</td>
<td>184K</td>
<td>[May, Aug)</td>
</tr>
<tr>
<td>r3</td>
<td>Jud</td>
<td>M</td>
<td>153K</td>
<td>[Apr, Sep)</td>
</tr>
</tbody>
</table>

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

▶ Output: Temporal (at each point in time) aggregation result

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>B</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>z1</td>
<td>CS</td>
<td>89K</td>
<td>[Feb, May)</td>
</tr>
<tr>
<td>z2</td>
<td>CS</td>
<td>276K</td>
<td>[May, Aug)</td>
</tr>
<tr>
<td>z3</td>
<td>M</td>
<td>153K</td>
<td>[Apr, Sep)</td>
</tr>
</tbody>
</table>

What database system functionality is needed to get such results?
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

Support for **time travel** through flashback technology:

- WHERE AS OF TIMESTAMP
- SetValidTime

```sql
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

<table>
<thead>
<tr>
<th>Row</th>
<th>ID</th>
<th>Name</th>
<th>City</th>
<th>validfrom</th>
<th>validto</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>Tina</td>
<td>Berlin</td>
<td>2013-10-01 08:30</td>
<td>2014-01-10 10:00</td>
</tr>
<tr>
<td>2</td>
<td>1002</td>
<td>Philip</td>
<td>London</td>
<td>2013-10-25 11:30</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>1003</td>
<td>John</td>
<td>New York</td>
<td>2013-11-05 09:00</td>
<td>2014-01-10 10:00</td>
</tr>
<tr>
<td>4</td>
<td>1004</td>
<td>John</td>
<td>Miami</td>
<td>2014-01-10 10:00</td>
<td>?</td>
</tr>
</tbody>
</table>

- 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
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

support for temporal keys, time travel
no support for temporal queries
Temporal support added as of Teradata 13.10

- 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

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)
  - Period datatype and UDF functions on periods
  - Indexing via GiST index

- PostgreSQL release 9.2 (2012)
  - 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 $p$

<table>
<thead>
<tr>
<th></th>
<th>$N$</th>
<th>$B$</th>
<th>$D$</th>
<th>$U$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{1a}$</td>
<td>Tom</td>
<td>181K</td>
<td>CS</td>
<td>[Feb, Aug]</td>
<td>[Feb, May]</td>
</tr>
<tr>
<td>$r_{1b}$</td>
<td>Tom</td>
<td>181K</td>
<td>CS</td>
<td>[Feb, Aug]</td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>$r_2$</td>
<td>Pam</td>
<td>184K</td>
<td>CS</td>
<td>[May, Aug]</td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>$r_3$</td>
<td>Jud</td>
<td>153K</td>
<td>M</td>
<td>[Apr, Sep]</td>
<td>[Apr, Sep]</td>
</tr>
</tbody>
</table>

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

- **Result**: Temporal Aggregation $D \forall T \text{SUM}(\text{scale}(B))(p)$

<table>
<thead>
<tr>
<th>$D$</th>
<th>$\text{SUM}$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>CS 89K</td>
<td>[Feb, May]</td>
</tr>
<tr>
<td>$z_2$</td>
<td>CS 276K</td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>$z_3$</td>
<td>M 153K</td>
<td>[Apr, Sep]</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>N</th>
<th>D</th>
<th>B</th>
<th>T_S</th>
<th>T_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>M</td>
<td>10k</td>
<td>Jan</td>
<td>Dec</td>
</tr>
<tr>
<td>P2</td>
<td>CS</td>
<td>7k</td>
<td>Feb</td>
<td>Aug</td>
</tr>
<tr>
<td>P3</td>
<td>CS</td>
<td>5k</td>
<td>Jun</td>
<td>Dec</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>Cnt</th>
<th>T_S</th>
<th>T_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Jun</td>
<td>Aug</td>
</tr>
<tr>
<td>2</td>
<td>Feb</td>
<td>Jun</td>
</tr>
</tbody>
</table>

Counting procedure:

\[ p \] @Jan

\[ N \quad D \quad B \]
| P1 | M | 10k |

\[ p \] @Feb

\[ N \quad D \quad B \]
| P1 | M | 10k |
| P2 | CS| 7k  |

\[ p \] @Mar

\[ N \quad D \quad B \]
| P1 | M | 10k |
| P2 | CS| 7k  |
Property P2

An period carries more information than a set of points.

- The following relations are semantically different:

<table>
<thead>
<tr>
<th>Reagan</th>
<th>1981/1/20</th>
<th>1985/1/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reagan</td>
<td>1985/1/20</td>
<td>1989/1/20</td>
</tr>
</tbody>
</table>

  \[ \neq \]

  | 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
Sometimes information must change if the associated period changes.

Query: What is the available project funding from 2011 to 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 $P_1 + P_2 + P_3$ (sequenced semantics)
  - $P_1$ (period can be decomposed into points)
  - $P_2$ (period carries more information than points)
  - $P_3$ (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 $P_2$
  - Research has focused on property $P_1$
  - SQL:2011 gives property $P_2$ and property $P_1$ 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 (∼ 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

IBM
Oracle
Teradata
SAP
Microsoft
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:
  
  \[(CS, [\text{May, Aug}]) \cong (CS, [\text{May, Aug}]) \rightarrow \text{true} \quad \text{OK}\]
  
  \[(CS, [\text{Feb, May}]) \cong (CS, [\text{May, Aug}]) \rightarrow \text{false} \quad \text{OK}\]
  
  \[(CS, [\text{Feb, Aug}]) \cong (CS, [\text{May, Aug}]) \rightarrow \text{false} \quad \text{NOT OK}\]

- What we want from a database system: adjusted periods that can be treated as \textbf{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 $\phi$

- 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.
  - $\epsilon_U(p)$: adds a copy of the timestamp as new attribute $U$
- Flexible **user-defined functions** for scaling.
  - $\pi_{scale}(B)(p)$: 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.
    ⇒ **Temporal Splitter**
    Example: Aggregation
  - One input tuple contributes to more than one result tuple at each point in time.
    ⇒ **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: \( D \varTheta^T_{COUNT(P)}(p) \)

<table>
<thead>
<tr>
<th>N</th>
<th>D</th>
<th>P</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>CS</td>
<td>PrjX</td>
<td>Feb, May</td>
</tr>
<tr>
<td>Pam</td>
<td>CS</td>
<td>PrjY</td>
<td>May, Aug</td>
</tr>
<tr>
<td>Jud</td>
<td>M</td>
<td>PrjZ</td>
<td>Apr, Sep</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>D</th>
<th>P</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>1</td>
<td></td>
<td>Feb, May</td>
</tr>
<tr>
<td>CS</td>
<td>2</td>
<td></td>
<td>May, Aug</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td></td>
<td>Apr, Sep</td>
</tr>
</tbody>
</table>

Adjustment (disjoint)

Non-temporal aggregation

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

- Managers project budgets: \( m \Join^T m.D = p.D \ p \)

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:** \( \psi^T \rightarrow \{ N, \phi \} + \psi \)

<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_B (r) )</td>
<td>( \pi_{B,T} (N_B(r,r)) )</td>
</tr>
<tr>
<td>Aggregation ( B \vartheta^T_F (r) )</td>
<td>( B,T \vartheta_F(N_B(r,r)) )</td>
</tr>
<tr>
<td>Difference ( r -^T s )</td>
<td>( N_A(r,s) - 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>( \alpha(\phi_T(r,s) \bowtie_{r,T=s,T} \phi_T(s,r)) )</td>
</tr>
<tr>
<td>Inner Join ( r \bowtie^T_\theta s )</td>
<td>( \alpha(\phi_\theta(r,s) \bowtie_{\theta \wedge r,T=s,T} \phi_\theta(s,r)) )</td>
</tr>
<tr>
<td>Left O. Join ( r \bowtie^T_\theta s )</td>
<td>( \alpha(\phi_\theta(r,s) \bowtie_{\theta \wedge r,T=s,T} \phi_\theta(s,r)) )</td>
</tr>
<tr>
<td>Right O. Join ( r \bowtie^T_\theta s )</td>
<td>( \alpha(\phi_\theta(r,s) \bowtie_{\theta \wedge r,T=s,T} \phi_\theta(s,r)) )</td>
</tr>
<tr>
<td>Full O. Join ( r \bowtie^T_\theta s )</td>
<td>( \alpha(\phi_\theta(r,s) \bowtie_{\theta \wedge r,T=s,T} \phi_\theta(s,r)) )</td>
</tr>
<tr>
<td>Anti Join ( r \bowtie^T_\theta s )</td>
<td>( \phi_\theta(r,s) \bowtie_{\theta \wedge r,T=s,T} \phi_\theta(s,r) )</td>
</tr>
</tbody>
</table>

Constructing Sequenced Algebra Expressions

Query: \( D \vartheta^T SUM(scale(B))(p) \)

1. Timestamp propagation:
   \( D \vartheta^T SUM(scale(B))(\epsilon U(p)) \)

2. Timestamp substitution:
   \( D \vartheta^T SUM(scale(B,T,U))(\epsilon U(p)) \)

3. Temporal adjustment:
   \( p' \leftarrow \mathcal{N}_D(\epsilon U(p), \epsilon U(p)) \)

4. Nontemporal aggregation:
   \( D, T \vartheta^T SUM(scale(B,T,U))(p') \)
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:

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

Query: What is the largest $X$?

Input:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

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

$$\text{SELECT MAX}(X)$$
$$\text{FROM } r;$$

$$\begin{array}{|c|}
\hline
\text{X} \\
\hline
7 \\
\hline
\end{array}$$

Input:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$X$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
<td>[Feb, Aug)</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>[May, Aug)</td>
</tr>
</tbody>
</table>

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

$$\begin{array}{|c|c|}
\hline
\text{X} & \text{T} \\
\hline
5 & [Feb, May) \\
7 & [May, Aug) \\
\hline
\end{array}$$

$$\text{SELECT MAX}(X), \text{Ts}, \text{Te}$$
$$\text{FROM (} r \text{ NORMALIZE } r \text{ USING()} \text{) } r$$
$$\text{GROUP BY } \text{Ts, Te;}$$
Query Q2

Query: What is the largest $X$ per $Y$?

Input:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

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

$$ \text{SELECT MAX}(X), Y \text{ FROM } r \text{ GROUP BY } Y; $$

Input:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A</td>
<td></td>
<td>[Feb, Aug]</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td></td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td></td>
<td>[Apr, Sep]</td>
</tr>
</tbody>
</table>

$$ Y \vartheta^T_{\text{MAX}}(X)(r) = Y, T \vartheta_{\text{MAX}}(X)(\mathcal{N}_Y(r, r)) $$

$$ \text{SELECT MAX}(X), Y, Ts, Te \text{ FROM } (r \text{ NORMALIZE } r \text{ USING}(Y)) \ r \text{ GROUP BY } Y, Ts, Te; $$

<table>
<thead>
<tr>
<th>$r$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A</td>
<td></td>
<td>[Feb, May]</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td></td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td></td>
<td>[Apr, Sep]</td>
</tr>
</tbody>
</table>
Query Q3

Query: What is the tuple with largest X?

Input:

<table>
<thead>
<tr>
<th>r</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>B</td>
</tr>
</tbody>
</table>

Answer: $r \triangleright^\theta s = \phi_\theta(r, s) \triangleright^{\theta \land T = s \cdot T} \phi_\theta(s, r)$

Input:

<table>
<thead>
<tr>
<th>r</th>
<th>X</th>
<th>Y</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>A</td>
<td>[Feb, Aug]</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>A</td>
<td>[May, Aug]</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>B</td>
<td>[Apr, Sep]</td>
</tr>
</tbody>
</table>

Answer: $r \triangleright^T s = \phi_{...}(r, s) \triangleright^{... \land T = s \cdot T} \phi_{...}(s, r)$

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

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);

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A</td>
<td>[Feb, May]</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>[May, Aug]</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>[Aug, Sep]</td>
</tr>
</tbody>
</table>
Query Q4

Query: What is the project budget per department?

Input:

\[
\begin{array}{|c|c|c|}
\hline
N & B & D \\
\hline
\text{Tom} & 181K & \text{CS} \\
\text{Pam} & 184K & \text{CS} \\
\text{Jud} & 153K & \text{M} \\
\hline
\end{array}
\]

Answer: \( D \vartheta \text{SUM}(B)(r) \)

\[
\text{SELECT } D, \text{SUM}(B) \\
\text{FROM p} \\
\text{GROUP BY } D;
\]

\[
\begin{array}{|c|c|}
\hline
D & \text{SUM} \\
\hline
\text{CS} & 365K \\
\text{M} & 153K \\
\hline
\end{array}
\]

Input:

\[
\begin{array}{|c|c|c|}
\hline
N & B & D \\
\hline
\text{Tom} & 181K & \text{CS} \\
\text{Pam} & 184K & \text{CS} \\
\text{Jud} & 153K & \text{M} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
T & \\
\hline
\text{[Feb, Aug)} & \\
\text{[May, Aug)} & \\
\text{[Apr, Sep)} & \\
\hline
\end{array}
\]

Answer: \( D \vartheta \text{SUM}(\text{scale}(B))(p) \)

\[
\text{WITH p2 AS (SELECT Ts Us, Te Ue, * FROM p)} \\
\text{SELECT D, SUM(scale(B,Ts,Te,Us,Ue)), Ts, Te} \\
\text{FROM (p2 NORMALIZE p2 USING(D)) p3} \\
\text{GROUP BY D, Ts, Te;}
\]

\[
\begin{array}{|c|c|c|}
\hline
D & \text{SUM} \\
\hline
\text{CS} & 89K \\
\text{CS} & 276K \\
\text{M} & 153K \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
T & \\
\hline
\text{[Feb, May)} & \\
\text{[May, Aug)} & \\
\text{[Apr, Sep)} & \\
\hline
\end{array}
\]
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:

```sql
CREATE OR REPLACE FUNCTION scaleU(x FLOAT, ts_new DATE, te_new DATE, ts_old DATE, te_old DATE)
RETURNS FLOAT AS $$
BEGIN
    RETURN x \ast (te\_new - ts\_new) / (te\_old - ts\_old);
END; $$ LANGUAGE PLPGSQL;
```
Scaling Function for Seasonal Scaling

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

\[
scaleT(x, T_{\text{new}}, T_{\text{old}}) = x \cdot \frac{\int_{T_{\text{Snew}}}^{T_{\text{Enew}}} f(t) \cdot dt}{\int_{T_{\text{Sold}}}^{T_{\text{Eold}}} f(t) \cdot dt}.
\]

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

![Graph showing cost trend and time](image-url)
Comparison of Scaling Functions

Input:

<table>
<thead>
<tr>
<th>N</th>
<th>B</th>
<th>D</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>181K</td>
<td>CS</td>
<td>[Feb, Aug)</td>
</tr>
<tr>
<td>Pam</td>
<td>184K</td>
<td>CS</td>
<td>[May, Aug)</td>
</tr>
<tr>
<td>Jud</td>
<td>153K</td>
<td>M</td>
<td>[Apr, Sep)</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>D</th>
<th>SUM</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>89K</td>
<td>[Feb, May)</td>
</tr>
<tr>
<td>CS</td>
<td>276K</td>
<td>[May, Aug)</td>
</tr>
<tr>
<td>M</td>
<td>153K</td>
<td>[Apr, Sep)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>SUM</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>83.62K</td>
<td>[Feb, May)</td>
</tr>
<tr>
<td>CS</td>
<td>281.38K</td>
<td>[May, Aug)</td>
</tr>
<tr>
<td>M</td>
<td>153K</td>
<td>[Apr, Sep)</td>
</tr>
</tbody>
</table>
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


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 $P_1 + P_2 + P_3$
- **Two** primitives are required to add support for sequenced semantics to a DBMS.
  - Temporal Splitter $\mathcal{N}^{-}$
  - Temporal Aligner $\phi$
- 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 patterns) is interesting

Thank You!