Temporal Data Management: An Overview

Johann Gamper

Free University of Bozen-Bolzano, Italy

gamper@inf.unibz.it
www.inf.unibz.it/~gamper

Joint work with
Michael Böhlen, University of Zurich
Anton Dignös, Free University of Bozen-Bolzano
Christian Jensen, Aalborg University

Free University of Bozen-Bolzano

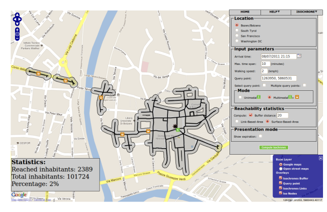
- Free University of Bozen-Bolzano
  - Founded in 1997
  - Faculties: Economics, Education, Computer Science, Design and Arts, Science and Technologies
  - Trilingual: English, Italian, German
- Faculty of Computer Science
  - Founded in 2001
  - Programmes: BSc, MSc, PhD
  - Teaching is mostly in English
  - Main research areas
    - Applied Software Engineering (CASE)
    - Information and Database Systems Engineering (IDSE)
    - Knowledge and Data (KRDB)

DB Research @ IDSE – Temporal Databases

- Interval-timestamped data in RDBMSs
  - More on this shortly
- Temporal aggregation algorithms
  - M. Böhlen, J. Gamper, C.S. Jensen: *Multi-dimensional aggregation for temporal data*, **EDBT 2006**
  - J. Gordevicius, J. Gamper, M.H. Böhlen: *Parsimonious temporal aggregation*, **The VLDB Journal 2012**
- Time series (imputation of missing values)
  - K. Wellenzohn, M. H. Böhlen, A. Dignös, J. Gamper, H Mitterer: Continuous imputation of missing values in streams of pattern-determining time series, **EDBT 2017**
  - M. Khayati, M.H. Böhlen, and J. Gamper: *Memory-efficient centroid decomposition for long time series*, **ICDE 2014**
- Complex event pattern matching
  - B. Cadonna, J. Gamper, M.H. Böhlen: *Sequenced event set pattern matching*, **EDBT 2011**

DB Research @ IDSE – Spatial Databases

- Isochrones and reachability analysis
  - J. Gamper, M.H. Böhlen, and M. Innerebner: *Scalable computation of isochrones with network expiration*, **SSDBM 2012**



- Itinerary and shortest path planning
  - T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser: Exact and approximate algorithms for finding k-shortest paths with limited overlap. **EDBT 2017**
  - P. Bolzoni, S. Helmer, K. Wellenzohn, J. Gamper, P. Andritsos: *Efficient itinerary planning with category constraints*, **SIGSPATIAL 2014**

DB Research @ IDSE – Graphs and Histograms

- D. Blumenthal, J. Gamper: Correcting and speeding-up bounds for non-uniform graph edit distance, **ICDE 2017**
- M. Shekelyan, A. Dignös, J. Gamper: DigitHist: a histogram-based data summary with tight error bounds, **PVLDB 2017**

Outline

1. Motivation
2. Background in Temporal Databases
3. History and State-of-the-Art
4. ANSI/ISO SQL:2011 Standard
5. SQL:2011 Standard in Commercial DBMSs
   - IBM DB2
   - Oracle Database
   - Teradata Database
   - PostgreSQL
6. Native Support for Sequenced Temporal Data
   - Application Requirements and Goal
   - Solution: Temporal Primitives
   - Mapping to SQL
   - PostgreSQL Implementation
   - Empirical Evaluation
7. Summary and Outlook

Outline

1. Motivation
2. Background in Temporal Databases
3. History and State-of-the-Art
4. ANSI/ISO SQL:2011 Standard
5. SQL:2011 Standard in Commercial DBMSs
6. Native Support for Sequenced Temporal Data
7. Summary and Outlook

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.*
  - *But no matter how big that data is or what insights we glean from it, it is still just a snapshot.*
  - *We need to stop getting stuck only on big data and start thinking about long data – datasets that have a massive historical sweep.*

- Increasing interest in temporal/historical data from big DB vendors

Long Data

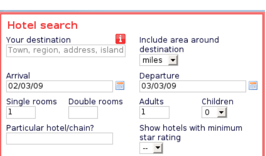
According to Benjamin Bruce from Pitney Bowes, businesses can enhance their customer relationships by embracing many of Arbesman's long data ideas [Jeff Bertolucci, InformationWeek 2013]:

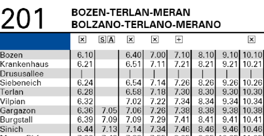
- *Big data is more about taking a slice in time across many different channels. But long data involves looking at information on a much longer timescale.*
- *Many companies have data that goes back 10, 20, 30 years. A longer-term view can provide information that businesses might miss if they examine 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.*

Temporal Data is Ubiquitous

- Almost all information is qualified with time (period or point)
  - Web, RDF stores (triples should really be quintuples)
  - Data warehousing
  - Medical records, loans, ...
  - Sensor data and time series
  - Transport information
  - ...

- Temporal data provide additional and more precise information for
  - Analysis
  - Prediction
  - Strategy planning
  - Accountability





201 BOZEN-TERLAN-MERAN / BOLZANO-TERLANO-MERANO

	☒	S A	☒	☒	+			☒
Bozen	6.10		6.40	7.00	7.10	8.10	9.10	10.10
Krankenhaus	6.21		6.51	7.11	7.21	8.21	9.21	10.21
Drususallee	|		|	|	|	|	|	|
Siebeneich	6.24		6.54	7.14	7.26	8.26	9.26	10.26
Terlan	6.28		6.58	7.18	7.30	8.30	9.30	10.30
Vilpian	6.32		7.02	7.22	7.34	8.34	9.34	10.34
Gargazon	6.36	7.05	7.06	7.26	7.38	8.38	9.38	10.38
Burgstall	6.39	7.09	7.09	7.29	7.41	8.41	9.41	10.41
Sinich	6.44	7.13	7.14	7.34	7.46	8.46	9.46	10.46
Meran Bhf.	7.00	7.40	7.33	7.55	8.05	9.05	10.05	11.05

The Need for Temporal Support in Databases

- Limited support for temporal data management in DBMSs
  - Conventional (non-temporal) DBs represent a static snapshot
  - Management of temporal aspects is implemented by the application
    - Adds additional complexity to application programs
  - Some time data types and functions are available in SQL, e.g., DATE, TIME, DATEADD(), DATEDIFF()
    - SQL:2011 added support for temporal tables
    - Still very limited query support
- A temporal database provides built-in support for the management of temporal data/time
  - Representation of various temporal aspects, e.g., valid time, transaction time
  - Support for multiple calendars and granularities
  - Easy formulation of complex queries over time
  - Queries over and modification of previous states
  - Temporal indeterminacy, including qualitative temporal relations

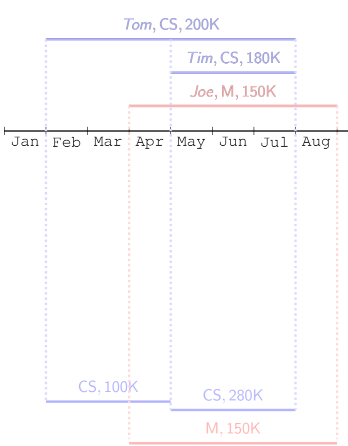
Example: Temporal Aggregation Query

- **Input**: External project funding

	N	D	B	T
r_1	Tom	CS	200K	[Feb, Jul]
r_2	Tim	CS	180K	[May, Jul]
r_3	Joe	M	150K	[Apr, Aug]

- **Query**: Amount of external funding per department?
- **Output**: Temporal (at each point in time) aggregation

	D	B	T
z_1	CS	100K	[Feb, Apr]
z_2	CS	280K	[May, Jul]
z_3	M	150K	[Apr, Aug]



Outline

1 Motivation

2 Background in Temporal Databases

3 History and State-of-the-Art

4 ANSI/ISO SQL:2011 Standard

5 SQL:2011 Standard in Commercial DBMSs

6 Native Support for Sequenced Temporal Data

7 Summary and Outlook

Time Domain/1

- Time domain/ontology
  - Specifies the basic building blocks of time
  - Time is generally modeled as an arbitrary set of instants/points with an imposed partial order, e.g., $(\mathbb{N}, <)$
  - Additional axioms introduce more refined models of time
- Structure of time
  - Linear time
    - Time advances from past to future in a step-by-step fashion
  - Branching time (possible future or hypothetical model)
    - Time is linear from the past to now, where it divides into several time lines
    - Along any future path, additional branches may exist
    - Structure is a tree rooted at now

Time Domain/2

- Density of time
  - Discrete time
    - Time instants are non-decomposable units of time with a positive duration, called chronons
    - Chronon is the smallest duration of time that can be represented
    - Isomorphic to natural numbers
  - Dense time
    - Between any two instants of time another instant exists
    - Isomorphic to rational numbers
  - Continuous time
    - Dense and no “gaps” between consecutive instants
    - Time instants are durationless
    - Isomorphic to the real numbers
- Boundness of time
  - Time can be bounded in the past and/or in the future, i.e., first and/or last time instant exists
- Relative (unanchored) versus absolute (anchored) time
  - “9 AM, January 1, 1996” is an absolute time
  - “9 hours” is a relative time

Time Domain/3

- Humans perceive time as continuous
- A discrete linear time model is generally used in temporal databases for several practical reasons:
  - Measures of time are generally reported in terms of chronons
  - Natural language references are compatible with chronons, e.g., 4:30 pm means over some period/chronon around this time
  - Chronons allow easily to model durative events
  - Any implementation needs a discrete encoding of time
- Problem to represent continuous movement in discrete model
  - An object is continuously moving from point A to point B

Time Granularity

- A time granularity is a partitioning of the timeline (chronons) into a finite set of segments, called granules.
- Provides a discrete image of a (possibly continuous) time
- Supports a user-friendly representation of time, e.g.,
  - birthdates are typically measured at granularity of days
  - business appointments to granularity of hours
  - train schedules to granularity of minutes
- Mixed granularities are of basic importance to modeling “real-world” temporal data
  - Converter functions between different granularities are needed

Temporal Data Models

- Data model: $M = (DS, QL)$
  - DS is a set of data structures
  - QL is a language for querying the data structures
- e.g., the relational data model is composed of relations and SQL
- Many extensions of the relational data model to support time have been proposed in past research
  - TSQL2, IXSQL, SQL/TP, ATSQL
- Several aspects have to be considered
  - Different time dimensions
  - Different timestamp types
  - Attribute versus tuple timestamping

Dimensions of Time/1

- Time is multi-dimensional
  - Valid time
  - Transaction time
  - Publication time
  - Efficacy time
  - etc.

Valid Time

- Valid time is the time a fact was/is/will be true in the modeled reality/mini-world
  - e.g., John has been hired on October 1, 2004
- Valid time captures the time-varying states of the modeled reality
- All facts have a valid time by definition, however, it might not be recorded in the database
- Valid time is independent of the recording of the fact in a database
- Can be bounded or unbounded

Transaction Time

- Transaction time is the time when a fact is current/present in the database as stored data
  - e.g., the fact "John was hired on October 1, 2004" was stored in the DB on October 5, 2004, and has been deleted on March 31, 2005
- Transaction time captures the time-varying states of the database
- Transaction time has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same fact
- Deletions of facts are purely logical
  - the fact remains in the database, but ceases to be part of the database's current state.
- Always bounded on both ends
  - Starts when the database is created (nothing was stored before)
  - Does not extend past now (no facts are known to have been stored in the future)
- Basis for supporting accountability and "traceability" requirements, e.g., in financial, medical, legal applications.
- Supplied automatically by the DBMS

Dimensions of Time/2

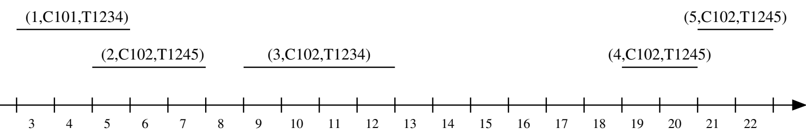
- A data model can support none, one, two, or more time dimensions
  - Snapshot data model: None of the time dimensions is supported
    - Represents a single snapshot of the reality and the database
  - Valid-time data model: Supports only valid time
  - Transaction-time data model: Supports only transaction time
  - Bitemporal data model: Supports valid time and transaction time

Timestamps/1

- Timestamp: A time value that is associated with an attribute value or a tuple in a database
  - Captures some temporal aspect, e.g., valid time, transaction time
  - Represented as one or more attributes/columns of a relation
- Three different types of timestamps are widley used
  - Time points
  - Time intervals
  - Temporal elements
- Two different ways of timestamping
  - Tuple timestamping
  - Attribute timestamping

Timestamps/2

- Example: Video store where customers, identified by a CustID, rent video tapes, identified by a TapeNum. Consider the following rentals during May 1997:
  - On 3rd of May, customer C101 rents tape T1234 for three days
  - On 5th of May, customer C102 rents tape T1245 for 3 days
  - From 9th to 12th of May, customer C102 rents tape T1234
  - From 19th to 20th of May, and again from 21st to 22nd of May, customer C102 rents tape T1245
- These rentals are stored in a relation Checkout which is graphically illustrated below



Point-Based Data Models

- Point-based data model: each tuple is timestamped with a time point/instant
- Most basic and simple data model
- Timestamps are atomic values and can be easily compared with $=, \neq, <, >, \geq, \leq$
- Syntactically different relations store different information
- Multiple tuples are used if a fact is valid at several time points
- Additional attributes required to restore original relation (SeqNo for 2-day rentals)
- Abstract view of a database and not meant for physical implementation
- Simplicity and computational complexity make it popular for theoretical studies

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	3
1	C101	T1234	4
1	C101	T1234	5
2	C102	T1245	5
2	C102	T1245	6
2	C102	T1245	7
3	C102	T1234	9
3	C102	T1234	10
3	C102	T1234	11
3	C102	T1234	12
4	C102	T1245	19
4	C102	T1245	20
5	C102	T1245	21
5	C102	T1245	22

Interval-Based Data Models

- Each tuple is timestamped with a time interval (time period)

CheckOut

SeqNo	CustID	TapeNum	T
1	C101	T1234	[3,5]
2	C102	T1245	[5,7]
3	C102	T1234	[9,12]
4	C102	T1245	[19,20]
5	C102	T1245	[21,22]

- Timestamps are atomic values that can be compared using Allen's 13 basic relationships between intervals (before, meets, during, etc.)
  - more convenient than comparing the endpoints of the intervals
- Multiple tuples are used if a fact is valid over disjoint time intervals
- `SeqNo` attribute is not needed to distinguish between different tuples
- The most popular model from an implementation perspective
- Time intervals are not closed under all set operations
  - e.g., subtracting $[5,7]$ from $[1,9]$ gives a set of intervals $\{[1,4],[8,9]\}$

Weak Interval-Based Data Models

- Intervals are only used as a convenient representation of (contiguous) sets of time points
- The following two relations are considered equivalent

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,22]

- The relations are (syntactically) different, but snapshot equivalent, i.e., they contain the same snapshots
- Semantically, a point-based view is adopted
- Coalescing of relations makes sense
  - Syntactically different relations are also semantically different
  - Impossible to distinguish two consecutive 2-day checkouts without an additional attribute

Strong Interval-Based Data Models

- Intervals are atomic units (and not just sets of time points) and carry meaning/semantics
- The following relations are considered to contain different information

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,20]
C102	T1245	[21,22]

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7]
C102	T1234	[9,12]
C102	T1245	[19,22]

- There is a difference between one 4-day checkout and two consecutive 2-day checkouts (e.g., different fees)
- Inherently interval-based models do not enforce
- More expressive: Can distinguish between a 4-day checkout and two consecutive 2-day checkouts without an additional attribute

Data Models with Temporal Elements

- Each tuple is timestamped with a temporal element, i.e., a finite union of intervals

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7] $\cup$ [19,20] $\cup$ [21,22]
C102	T1234	[9,12]

CheckOut

CustID	TapeNum	T
C101	T1234	[3,5]
C102	T1245	[5,7] $\cup$ [19,22]
C102	T1234	[9,12]

- The full history of a fact is stored in one tuple
- Point-based and interval-based semantics are possible (as for the interval-based data model)
  - Under the point-based view, both relations are equivalent

Attribute Value Timestamping/1

- Each attribute value is timestamped with a set of points/intervals
- Captures all information about a real-world object in a single tuple
  - e.g., all information about a customer in the relation below
- Information about other objects is spread across several tuples, e.g., information about tapes.

CheckOut

SeqNo		CustID		TapeNum	
[3,5]	1	[3,5]	C101	[3,5]	T1234
[5,7]	2	$[5,7] \cup [9,12] \cup [19,22]$	C102	$[5,7] \cup [19,22]$	T1245
[9,12]	3			[9,12]	T1234
[19,20]	4				
[21,22]	5				

- A single tuple may record multiple facts
  - second tuple records the facts: rental information for customer `C102` for the tapes `T1245` and `T1234`, and four different checkouts
- Non-first-normal-form data model

Attribute Value Timestamping/2

- Different groupings of the information into tuples are possible for attribute-value timestamping
  - e.g., grouping on tape number in the example below

CheckOut

SeqNo		CustID		TapeNum	
[3,5]	1	[3,5]	C101	[3,5] ∪ [9,12]	T1234
[9,12]	3	[9,12]	C102		
[5,7]	2	[5,7] ∪ [19,22]	C102	[5,7] ∪ [19,22]	T1245
[19,20]	4				
[21,22]	5				

- This relation is snapshot-equivalent to the previous one (grouped on customers).

Outline

1. Motivation
2. Background in Temporal Databases
3. **History and State-of-the-Art**
4. ANSI/ISO SQL:2011 Standard
5. SQL:2011 Standard in Commercial DBMSs
6. Native Support for Sequenced Temporal Data
7. Summary and Outlook

Temporal Database Research History

- Four overlapping phases
  - 1956–1985: Concept development, considering the multiple kinds of time and conceptual modeling
  - 1978–1994: Design of query languages
    - 1978-1990: Relational temporal query languages
    - 1990–1994: Object-oriented temporal query langauges
  - 1988–present: Implementation aspects, including storage structures, operator algorithms, and temporal indexes.
  - 1993–present: Consolidation phase
    - Consensus glossary of temporal database concepts http://www.cs.aau.dk/˜csj/Glossary/index.html
    - Query language test suite
    - TSQL2
- Still an active research area today
  - New application domains with the need for new operations
    - Spatio-temporal and moving-object databases, e.g., mobile-phone tracking to monitor employees, company cars, and equipment
    - Data streams
    - Data warehousing

State-of-the-Art – Indexing

- Martin Kaufmann, Peter M. Fischer, Norman May, Chang Ge, Anil K. Goel, Donald Kossmann: Bi-temporal Timeline Index: A data structure for Processing Queries on bi-temporal data. ICDE 2015: 471-482
- Martin Kaufmann: Storing and Processing Temporal Data in a Main Memory Column Store. PVLDB 6(12): 1444-1449 (2013)
- Martin Kaufmann, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber: Comprehensive and Interactive Temporal Query Processing with SAP HANA. PVLDB 6(12): 1210-1213 (2013)
- Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber, Norman May: Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. SIGMOD Conference 2013: 1173-1184

State-of-the-Art – Joins

- D. Gao, C.S. Jensen, R.T. Snodgrass, M.D. Soo: Join operations in temporal databases. VLDB Journal, vol. 14, 2005.
- Francesco Cafagna, Michael H. Böhlen: Disjoint interval partitioning. VLDB Journal, vol. 26, 2017.
- Danila Piatov, Sven Helmer, Anton Dignös: An interval join optimized for modern hardware. Proc. of ICDE, 2016.
- Anton Dignös, Michael H. Böhlen, Johann Gamper: Overlap interval partition join. Proc. of SIGMOD, 2014.

State-of-the-Art – Aggregation

- N. Klein, R.T. Snodgrass: Computing temporal aggregates. Proc. of ICDE 1995.
- B. Moon, I.F. Vega Lopez, V. Immanuel: Efficient algorithms for large-scale temporal aggregation. TKDE, vol. 15, 2004.
- Esteban Zimányi: Temporal aggregates and temporal universal quantification in standard SQL. SIGMOD Record, vol. 35, 2006.
- M. Böhlen, J. Gamper, C.S. Jensen: Multi-dimensional aggregation for temporal data. Proc. of EDBT, 2006.
- J. Gordevicius, J. Gamper, M. Böhlen: Parsimonious temporal aggregation. Proc. of EDBT, 2009.
- J. Gordevicius, J. Gamper, M.H. Böhlen. Parsimonious temporal aggregation. VLDB Journal, 2012.

State-of-the-Art – Relational Databases

- Anton Dignös, Michael H. Böhlen, Johann Gamper, Christian S. Jensen: Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries. TODS, vol. 41, 2016.
- Andreas Behrend, Philip Schmiegelt, Jingquan Xie, Ronny Fehling, Adel Ghoneimy, Zhen Hua Liu, Eric S. Chan, Dieter Gawlick: Temporal state management for supporting the real-time analysis of clinical data. Proc. of ADBIS, 2014.
- Anton Dignös, Michael H. Böhlen, Johann Gamper: Query time scaling of attribute values in interval timestamped databases. Proc. of ICDE, 2013.

State-of-the-Art – Query Languages

- M. Böhlen, C.S. Jensen: Temporal data model and query language concepts. Encyclopedia of Information Systems, vol. 4, 2003.
- R. T. Snodgrass, editor. The TSQL2 Temporal Query Language. Kluwer, 1995.
- R. T. Snodgrass. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, Inc., San Francisco, CA, July 1999.
- M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass, *Temporal Statement Modifiers*, TODS 2000.
- 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.

State-of-the-Art – Misc

- Temporal probabilistic databases
  - Katerina Papaioannou, Michael H. Böhlen: TemProRA: Top-k temporal-probabilistic results analysis. Proc. of ICDE, 2016.
- Temporally evolving graphs
  - Vera Zaychik Moffitt, Julia Stoyanovich: Towards sequenced semantics for evolving graphs. Proc. of EDBT, 2017.
  - Vera Zaychik Moffitt, Julia Stoyanovich: Portal: A Query Language for Evolving Graphs. CoRR abs/1602.00773 (2016)
- Semantics
  - Curtis E. Dyreson, Ventkata A. Rani, Amani Shatnawi: Unifying Sequenced and Non-sequenced Semantics. TIME 2015: 38-46

Outline

1. Motivation
2. Background in Temporal Databases
3. History and State-of-the-Art
4. ANSI/ISO SQL:2011 Standard
5. SQL:2011 Standard in Commercial DBMSs
6. Native Support for Sequenced Temporal Data
7. Summary and Outlook

Running Example

- Employees assigned to departments

Emp

EName	EStart	EEnd	EDept
Anton	2010	2015	ifi
Anton	2015	2017	idse

- Departments with a description

Dept

DName	DStart	DEnd	DDesc
ifi	2009	9999	DBTG @ uzh
idse	2010	2016	DB @ unibz
idse	2015	9999	DBS @ unibz

- In SQL, the start/end points are of type DATE, e.g., `2010-01-01`
- To simplify illustration, most examples use only the year

ANSI/ISO SQL:2011 Standard – Support for Temporal Data

- Most important new functionality of SQL:2011 is the ability to create and manipulate temporal tables
- Key features
  - Period specification
  - Application-time/System-time period tables
  - Temporal UPDATE/DELETE behavior
  - Temporal key constraints
  - Predicates and functions for periods
- Krishna Kulkarni and Jan-Eike Michels. Temporal features in SQL:2011. SIGMOD Rec. 41, 3 (October 2012), pp 34-43, 2012.
- A previous attempt to include temporal support (i.e., TSQL2) in SQL was not successful

SQL:2011: Period Specification

- SQL:2011 adds period definitions as metadata to tables (and not as a new data type!)
- A period is a conceptual grouping of a physical start time and end time attribute/column
- For instance, a period `EPeriod` built from existing attributes `EStart` and `EEnd`

```
... PERIOD FOR EPeriod (EStart, EEnd) ...
```

- A period is by default a half-closed interval `[Estart, EEnd)`
- Distinction between
  - application time (= valid time)
  - system time (= transaction time)
- By designing periods as metadata, backward compatibility is achieved: old schemas, queries and tools still work

Application-time Period Tables

- Create an application-time period table

```
CREATE TABLE Emp (
  EName VARCHAR,
  EStart DATE,
  EEnd DATE,
  EDept VARCHAR,
  PERIOD FOR EPeriod (EStart, EEnd)
);
```

- The user specifies the application time of the tuples

```
INSERT INTO Emp VALUES (Anton, 2010, 2015, ifi);
```

Emp

EName	EStart	EEnd	EDept
Anton	2010	2015	ifi

Application-time UPDATE/DELETE Behavior

- Conventional UPDATE/DELETE works
- Enhanced support to modify tuples over parts of period
  - Overlapping tuples are automatically split/cut

Emp

EName	EStart	EEnd	EDept
Anton	2010	2015	ifi

```
DELETE Emp
  FOR PORTION OF EPeriod
    FROM DATE '2011' TO DATE '2013'
WHERE EName = 'Anton'
```

Emp

EName	EStart	EEnd	EDept
Anton	2010	2011	ifi
Anton	2013	2015	ifi

Application-time Primary Keys

- Primary keys typically require to include the application-time period
  - Only one value at a time is allowed for non-temporal key attributes
  - In other words, application-time period has to be disjoint for different values of the non-temporal key attributes
  - e.g., an employee is never in two different depts at the same time

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

- The following table would not be allowed

Emp

EName	EStart	EEnd	EDept
Anton	2010	2015	ifi
Anton	2014	9999	idse

- Without OVERLAPS, it would be a conventional primary key

Application-time Foreign Keys

- Foreign keys guarantee that at each time point a corresponding row exists in the referenced table
  - e.g., each employee works at any time in an existing department

```
ALTER TABLE Emp
ADD FOREIGN KEY (EDept, PERIOD EPeriod)
  REFERENCES (DName, PERIOD DPeriod)
```

Emp

EName	EStart	EEnd	EDept
Anton	2010	2015	ifi
Anton	2014	9999	idse

Dept

DName	DStart	DEnd	DDesc
ifi	2009	9999	DBTG @ uzh
idse	2010	2016	DB @ unibz
idse	2016	9999	DBS @ unibz

System-time Period Tables/1

- System-time periods record when a tuple is known to the database
  - End points of type DATE or TIMESTAMP
- Periods are generated by the DBMS upon INSERT/UPDATE, using the current timestamp (e.g., 2015)

```
CREATE TABLE Dept (
  DName VARCHAR,
  DStart DATE GENERATED ALWAYS AS ROW START,
  DEnd DATE GENERATED ALWAYS AS ROW END,
  DDesc TEXT,
  PERIOD FOR SYSTEM_TIME (DStart, DEnd)
) WITH SYSTEM VERSIONING;

INSERT INTO DEPT (DName, DDesc)
  VALUES ('idse', 'DBS @ unibz')
```

Dept

DName	DStart	DEnd	DDesc
idse	2015	9999	DBS @ unibz

System-time Period Tables/2

- User is not allowed to change the system-time period; this is done by the DBMS
- Inserting a tuple sets the start time to the current transaction time
- System-time period is automatically modified when non-temporal attributes are modified
- A tuple is considered as current system tuple if the timestamp contains the current time (aka now)
- All other tuples are called historical tuples
- Historical tuples are never modified; they form immutable snapshots of the past

System-time UPDATE/DELETE Behavior

- UPDATE operates only on current tuples and automatically
  - inserts a historical tuple with end time = current time
  - inserts a current tuple with start time = current time
- DELETE creates only a historical tuple with end time = current time

Dept

DName	DStart	DEnd	DDesc
idse	2010	9999	DB @ unibz

```
UPDATE Dept
SET DDesc = 'DBS @ unibz'
WHERE DName = 'idse'
```

Dept

DName	DStart	DEnd	DDesc
idse	2010	2015	DB @ unibz
idse	2015	9999	DBS @ unibz

System-time Key Constraints

- Enforcement of primary key and foreign key contraints is much simpler as only current tuples need to be considered
  - Historical data continue to satisfy the constraints
- System-time period need not to be included in the definition of keys
- Primary key that ensures that there exists at most one current tuple with a given DName

```
ADD PRIMARY KEY (DName)
```

- Hence, never two tuples with the same name at the same time
- Foreign key that enforces that a dept with DName = EDept needs to exist NOW

```
ADD FOREIGN KEY (EDept) REFERENCES (DName)
```

Bitemporal Tables

- Tables can be both system-versioned and having an appilcation-time period
- Aka bitemporal tables that record when a fact is true in the modeled reality and when the fact was recorded in the database
- At most one system-time period and one application-time period per table

Application-time Queries

- Using the usual SQL syntax with constraints on the period end points
- Period predicates to facilitate query formulation, e.g., OVERLAPS, BEFORE, AFTER, etc.
  - Do not exactly correspond to Allen's interval relations
- Retrieve employees who worked as of January 2, 2011

```
SELECT *
FROM Emp
WHERE EPeriod CONTAINS DATE '2011-01-02'
```

- Temporal join of Emp and Dept relations

```
SELECT *
FROM Emp JOIN Dept
     ON EDept = DName
     AND EPeriod OVERLAPS DPeriod
```

System-time Queries

- Three SQL extensions to retrieve data
- Retrieve tuples as of a given time point (i.e., tuples with start time less or equal and end time greater)

```
SELECT *
FROM Emp FOR SYSTEM_TIME AS OF DATE '2010-01-02'
```

- Retrieve tuples between any two points in time
  - End time is not included

```
SELECT *
FROM Dept FOR SYSTEM_TIME
     FROM DATE '2011-01-01' TO DATE '2012-01-01'
```

  - End time is included

```
SELECT *
FROM Dept FOR SYSTEM_TIME
     BETWEEN DATE '2011-01-01' AND DATE '2011-12-31'
```

- If nothing is specified, only current tuples are considered

Summary of SQL:2011

- Periods are closed-open intervals $[X, Y)$
- Periods are not explicit attributes, but specified as additional schema information
  - Backward compatible to two columns!
- Tables can have at most one sytem-time period and one appilcation-time period
- No concept of "NOW", "UC", "infinity", "empty"
  - Only 0001-01-01 and 9999-12-31 are "special" dates
- Many differences between application and system time period tables
- Limited support for query formulation

Outline

1. Motivation
2. Background in Temporal Databases
3. History and State-of-the-Art
4. ANSI/ISO SQL:2011 Standard
5. **SQL:2011 Standard in Commercial DBMSs**
6. Native Support for Sequenced Temporal Data
7. Summary and Outlook

IBM DB2/1

- Key features: fully supports SQL:2011
  - Period specification
  - Application-time/System-time period tables
  - Temporal UPDATE/DELETE behavior
  - Temporal key constraints
  - Predicates and functions for periods

- A matter of time: Temporal data management in DB2 10
  `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf`

IBM DB2/2

- Implementation of system-time with current and history table

Current Dept

DName	DStart	DEnd	DDesc
ifi	2009	9999	DBTG @ uzh
idse	2015	9999	DBS @ unibz

History Dept

DName	DStart	DEnd	DDesc
idse	2010	2015	DB @ unibz

- Operations on system-time tables only affect the current state/table
  - Temporal UPDATE/DELETE behavior
  - Temporal key constraints

Oracle Database

- Key features
  - Periods (stored in a single column)
  - Valid-time/transaction-time tables
  - Temporal UPDATE/DELETE behavior
  - Temporal key constraints
  - Predicates and functions for periods

- Implemented in Workspace Database Manager

- C. Murray. Oracle database workspace manager developer's guide. `http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28396.pdf`

Valid-time Tables

```
CREATE TABLE Dept (
  DName VARCHAR
  DDesc TEXT
);
EXECUTE DBMS_WM.EnableVersioning ('Dept');
EXECUTE DBMS_WM.AlterVersionedTable('Dept', 'ADD_VALID_TIME');

INSERT INTO Dept VALUES
  ('idse', 'DB @ unibz',
   WMSYS.WM_PERIOD(TO_DATE('2010-01-01', 'YYYY-MM-DD'),
                   TO_DATE('9999-01-01', 'YYYY-MM-DD')));
```

Dept

DName	DDesc	WM_VALID(VALIDFROM, VALIDTILL)
idse	DB @ unibz	WM_PERIOD('2010-01-01', '9999-01-01')

Valid-time UPDATE/DELETE Behavior

- UPDATE/DELETE are executed with the current session's valid time
- DBMS_WM.SetValidTime sets the valid time
  - If not specified, current time is the default

```
EXECUTE DBMS_WM.SetValidTime(
  TO_DATE('2015-02-05', 'YYYY-MM-DD'),
  TO_DATE('9999-01-01', 'YYYY-MM-DD'));

UPDATE Dept
SET DDesc = 'DBS @ unibz'
WHERE DName = 'idse';
```

Dept

DName	DDesc	WM_VALID(VALIDFROM, VALIDTILL)
idse	DB @ unibz	WM_PERIOD('2010-01-01', '2015-02-05')
idse	DBS @ unibz	WM_PERIOD('2015-02-05', '9999-01-01')

Queries

- Similar to UPDATE/DELETE, queries are issued against the current session's valid time
- Period needs to be explicitly specified in SELECT to be visible

```
SELECT * FROM Dept;
```

Dept

DName	DDesc
idse	DBS @ unibz

```
SELECT DName, DDesc,
       TO_CHAR(valid_time_start,'YYYY-MM-DD') "DStart",
       TO_CHAR(valid_time_end,'YYYY-MM-DD') "DEnd" FROM Dept;
```

Dept

DName	DDesc	DStart	DEnd
idse	DBS @ unibz	'2015-02-05'	'9999-01-01'

Summary of Oracle Database

- UPDATE/DELETE/SELECT are on current data/valid time
- Period columns are implicit
- "Versioning" can be enabled and disabled (default)
- Two time dimensions possible (valid and transaction time)

Teradata Database

- Key features
  - Periods
  - Temporal UPDATE/DELETE behavior
  - Temporal key constraints
  - Predicates and functions for periods
  - Queries with temporal statement modifiers

- Currently DBMS with most temporal features!

- Teradata database temporal table support. http://www.info.teradata.com/edownload.cfm?itemid=102320064

Create Temporal Tables

```
CREATE MULTISET TABLE Dept (
  DName VARCHAR
  DPeriod PERIOD(DATE) AS VALIDTIME
  DDesc TEXT,
);
```

- Statement modifiers are used to manipulate temporal tables
  - Specifiy the semantics of the SQL statement

```
SEQUENCED VALIDTIME
INSERT INTO Dept VALUES
  ('idse', 'DB @ unibz',
   PERIOD (DATE '2010', DATE '9999'));
```

Dept

DName	EPeriod	DDesc
idse	PERIOD('2010', '9999')	DB @ unibz

Temporal UPDATE/DELETE

Dept

DName	EPeriod	DDesc
idse	PERIOD('2010', '9999')	DB @ unibz

```
SEQUENCED VALIDTIME PERIOD '(2015, 9999)'
UPDATE Dept
SET DDesc = 'DBS @ unibz'
WHERE DName = 'idse'
```

Dept

DName	EPeriod	DDesc
idse	PERIOD('2010', '2015')	DB @ unibz
idse	PERIOD('2015', '9999')	DBS @ unibz

Temporal Key Constraints

```
CREATE TABLE Emp (
  EName VARCHAR SEQUENCED VALIDTIME PRIMARY KEY
  EPeriod PERIOD(DATE) AS VALIDTIME
  EDept TEXT,
  FOREIGN KEY (EDept) REFERENCES (DName)
);
```

- Primary key constraints are checked by the database
- But referential constraints are not enforced by the database*
  - Must be checked by the user
  - If specified, the database can use them to improve query performance
  - Helpful for a good design

*Teradata database temporal table support.
http://www.info.teradata.com/edownload.cfm?itemid=102320064, pp. 215

Teradata Statement Modifiers

- Teradata has many statement modifiers with different behaviors (we have only seen SEQUENCED VALIDTIME)

```
CURRENT VALIDTIME

VALIDTIME AS OF

SEQUENCED VALIDTIME

NONSEQUENCED VALIDTIME
```

- The same modifiers for TRANSACTIONTIME

- Example: Change the description of department from now on

```
CURRENT VALIDTIME
UPDATE Dept
SET DDesc = 'DBS @ unibz'
WHERE DName = 'idse'
```

Sequenced Valid-time Queries

- The sequenced form of a temporal query is limited to a simple SELECT from a single valid-time table or a simple SELECT with inner joins from multiple tables.
- A non-correlated scalar subquery can be used in the temporal query.
- The following operations are not supported for sequenced queries:
  - Outer joins
  - Set operations
  - Ordered analytic functions
  - Subqueries other than non-correlated scalar subqueries
  - `WITH`, `WITH RECURSIVE`, `TOP n`, `GROUP BY` or `DISTINCT`

PostgreSQL

- Key features:
  - Range types (Periods)
  - Indexes on range types
  - Temporal constraints using indexes
  - Predicates and functions for range types

- http://www.postgresql.org/docs/devel/static/rangetypes.html

Tables with Range Types

- A new data type DATARANGE to represent (time) intervals

```
CREATE TABLE Emp (
  EName VARCHAR
  EPeriod DATERANGE,
  EDept VARCHAR,
);
```

```
INSERT INTO Emp VALUES ('Anton', '[2010, 2014)', ifi);
```

Emp

EName	EPeriod	EDept
Anton	'[2010, 2014)'	ifi

Key Constraints

- Primary keys are achieved through a GiST index
- Enforces that an attibute has to have the same value (=) over overlapping time periods (&&)

```
CREATE TABLE Emp (
  EName VARCHAR
  EPeriod DATERANGE,
  EDept VARCHAR,
)

EXCLUDE USING GIST (EName WITH =, EPeriod WITH &&)
```

- Foreign keys are not directly supported (e.g., by using the GiST index as for primary keys)
- Triggers could be used

Indexes on Range Types

- Provides a powerful index: General inverted search tree (GiST)
  - On periods and combinations of period and other attributes

```
CREATE INDEX Emp_idx ON Emp
  USING GIST (EPeriod)

CREATE INDEX Emp_idx ON Emp
  USING GIST (EName, EPeriod)
```

- Improves range indexing for many predicates
  e.g., EQUAL, OVERLAP, LESS THAN, CONTAINS, ...

Queries

- SQL with predicates, e.g., OVERLAPS (&&), BEFORE, AFTER

```
SELECT *
FROM Emp JOIN Dept
     ON EPeriod && DPeriod

SELECT *
FROM Dept
WHERE upper_inf(DPeriod) = TRUE
```

- Some additional functions on ranges: UNION (+), INTERSECTION (*), DIFFERENCE (-)

```
SELECT '[2010, 2013)' + '[2012, 2015)'
```

Summary of PostgreSQL

- Flexible range types
- Many predicates and functions
- A range boundary is not a datatype
- A range type value may be any kind of interval: [], (), [)
  What does `SELECT upper(EPeriod) FROM Emp` return?
- Additional functions (+, *, -) are of little use since they may return two ranges and thus throw an error

Summary Temporal Support in DBMSs/1

DBS	Period	Keys	Update	Predicates	Queries
SQL:2011	✓	✓	✓	✓	✗
IBM DB2	✓	✓	✓	✓	✗
MySQL	✗	✗	✗	✗	✗
MS SQL Server	✗	✗	✗	✗	✗
PostgreSQL	✓	~	✗	✓	✗
Oracle	✓	✓	✓	✓	✗
Teradata	✓	✓	✓	✓	~

Summary Temporal Support in DBMSs/2

- Many DBSs have temporal features, following SQL:2011 in some way or another
  - different syntax
  - different number of features
- Infrastructure is well settled
  - Period datatype
  - Key constraints
  - Simply SQL with predicates and functions

- 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
- **More support for queries and query processing is required!**

Outline

1. Motivation
2. Background in Temporal Databases
3. History and State-of-the-Art
4. ANSI/ISO SQL:2011 Standard
5. SQL:2011 Standard in Commercial DBMSs
6. **Native Support for Sequenced Temporal Data**
7. Summary and Outlook

Key Question

What core functionality should a DBMS offer to support the management and querying of data qualified with a time period?

or: what would be our wish from

Temporal Aggregation Example

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

p

	N	D	B	T
r_1	Tom	CS	200K	[Feb, Jul]
r_2	Tim	CS	180K	[May, Jul]
r_3	Joe	M	150K	[Apr, Aug]

- **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	100K	[Feb, Apr]
z_2	CS	280K	[May, Jul]
z_3	M	150K	[Apr, Aug]

- Timestamps must be adjusted for the result
- Some values must be scaled to the adjusted timestamps

Property P1

Property P1

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

- aka snapshot reducibility: $\tau_t(\psi^T(\mathbf{r})) \equiv \psi(\tau_t(\mathbf{r}))$

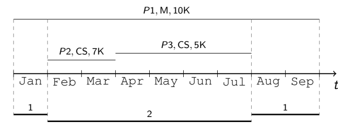
- **Query**: What are the top-2 time periods with most projects?

p

P	*D*	*B*	*T*
P1	M	10K	[Jan,Sep]
P2	CS	7K	[Feb,Mar]
P3	CS	5K	[Apr,Jul]

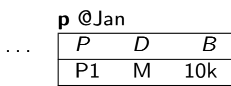
result

Cnt	*T*
1	[Jan,Jan]
2	[Feb,Jul]
1	[Aug,Sep]



- **Counting procedure**: count at each point in time (+ coalescing)

...


p @Jan

P	*D*	*B*
P1	M	10k

p @Feb

P	*D*	*B*
P1	M	10k
P2	CS	7k

p @Mar

P	*D*	*B*
P1	M	10k
P2	CS	7k

...

Property P2

Property P2

An period carries more information than a set of points.

- The following two relations are semantically different:

President	T
Reagan	[1981/1/20,1985/1/20]
Reagan	[1985/1/20,1989/1/20]

President	T
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/coalescing should be done conservatively
  - Only when requested by the application

Property P3

Property P3

Sometimes information must change if the associated period changes.

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

Proj	Budget	T
P1	300K	[2011,2014]

Proj	Budget	T
P1	150K	[2011,2012]

- We need a mechanism that allows applications to change values if the associated period is changed.

Goal

- We want a full-fledged relational algebra + database system that support properties P1 + P2 + P3
  - P1: period can be decomposed into points
  - P2: period carries more information than points
  - P3: sometimes values must change along with associated periods
- It is easy to get some of these properties, but difficult to get all of them together.
  - Research has focused on P1
  - SQL gives P2
  - P3 is not supported at all

The Limitation of Relational Algebra and SQL

- SQL treats periods as atomic units, which causes problems.
- What we get from a database system:

  $(\mathrm{CS}, [\texttt{May}, \texttt{Jul}]) \stackrel{?}{=} (\mathrm{CS}, [\texttt{May}, \texttt{Jul}]) \rightarrow \mathit{true}$ OK

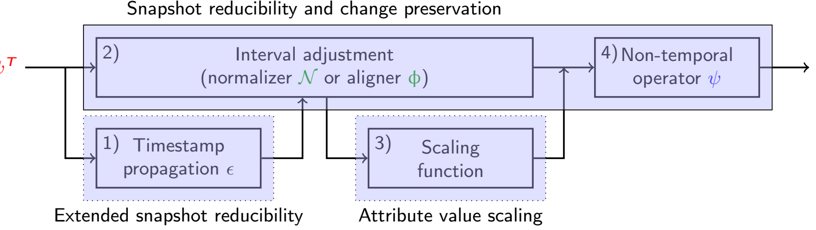
  $(\mathrm{CS}, [\texttt{Feb}, \texttt{Apr}]) \stackrel{?}{=} (\mathrm{CS}, [\texttt{May}, \texttt{Jul}]) \rightarrow \mathit{false}$ OK

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

- **Our approach:** adjust periods such that they can be treated as atomic units for which equality works again as expected.

Solution Overview

- Reduction of a temporal operator ψ^T to the corresponding nontemporal operator ψ



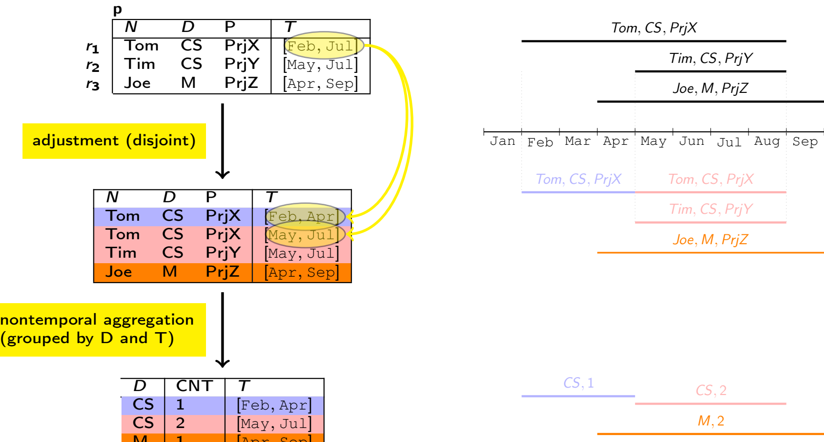
- Requires minimal changes to DBMS
  - Two new adjustment primitives are added to the kernel
  - Timestamp propagation is a projection
  - Attribute value scaling as user-defined functions
- Existing query optimization and indexing of DBMS can be used

Temporal Primitives

- Break timestamps into pieces that are aligned
- Then, equality on aligned timestamps can be used
- Two primitives are required:
  - Temporal Normalizer: for operators where one input tuple contributes to at most one result tuple at each point in time, e.g., aggregation
  - Temporal Aligner: for operators where one input tuple might contribute to more than one result tuple at each point in time, e.g., joins

Temporal Normalizer

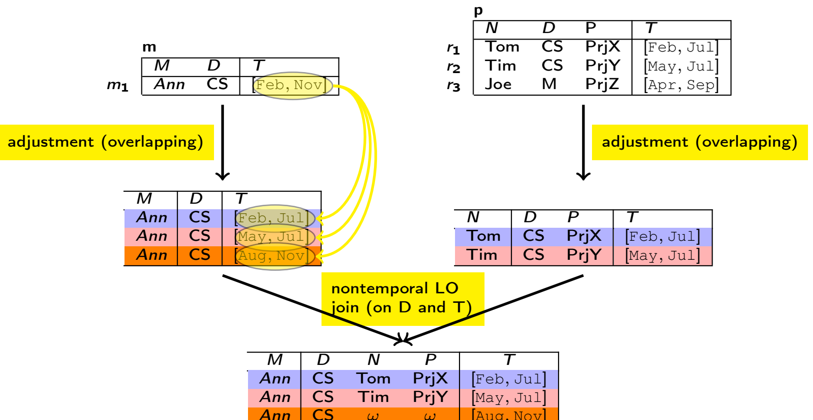
- Number of projects per department: ${}_D\vartheta^T_{COUNT(P)}(\mathbf{p})$



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

Temporal Aligner

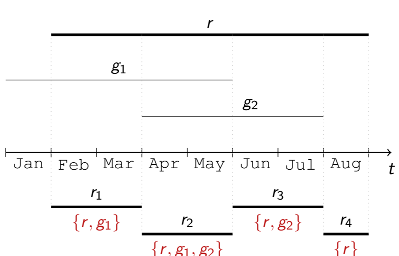
- Managers' project budgets: $\mathbf{m} \ ⟕^{T}_{\mathbf{m}.D=\mathbf{p}.D} \ \mathbf{p}$



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

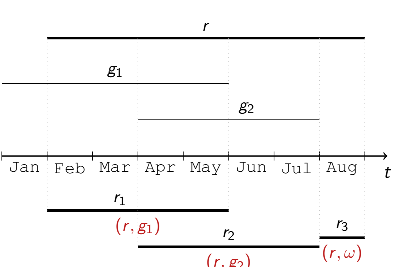
Temporal Normalizer vs. Temporal Aligner

Temporal Normalizer



- 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	$\sigma^T_\theta(\mathbf{r})$	$=$	$\sigma_\theta(\mathbf{r})$
Projection	$\pi^T_{\mathbf{B}}(\mathbf{r})$	$=$	$\pi_{\mathbf{B},T}(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$
Aggregation	${}_{\mathbf{B}}\vartheta^T_F(\mathbf{r})$	$=$	${}_{\mathbf{B},T}\vartheta_F(\mathcal{N}_{\mathbf{B}}(\mathbf{r},\mathbf{r}))$
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	$\mathbf{r} \bowtie^T_\theta \mathbf{s}$	$=$	$\alpha(\phi_\theta(\mathbf{r},\mathbf{s}) \bowtie_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T} \phi_\theta(\mathbf{s},\mathbf{r}))$
Left O. Join	$\mathbf{r}$ ⟕$^T_\theta$ $\mathbf{s}$	$=$	$\alpha(\phi_\theta(\mathbf{r},\mathbf{s})$ ⟕$_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T}$ $\phi_\theta(\mathbf{s},\mathbf{r}))$
Right O. Join	$\mathbf{r}$ ⟖$^T_\theta$ $\mathbf{s}$	$=$	$\alpha(\phi_\theta(\mathbf{r},\mathbf{s})$ ⟖$_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T}$ $\phi_\theta(\mathbf{s},\mathbf{r}))$
Full O. Join	$\mathbf{r}$ ⟗$^T_\theta$ $\mathbf{s}$	$=$	$\alpha(\phi_\theta(\mathbf{r},\mathbf{s})$ ⟗$_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T}$ $\phi_\theta(\mathbf{s},\mathbf{r}))$
Anti Join	$\mathbf{r} \triangleright^T_\theta \mathbf{s}$	$=$	$\phi_\theta(\mathbf{r},\mathbf{s}) \triangleright_{\theta\wedge\mathbf{r}.T=\mathbf{s}.T} \phi_\theta(\mathbf{s},\mathbf{r})$

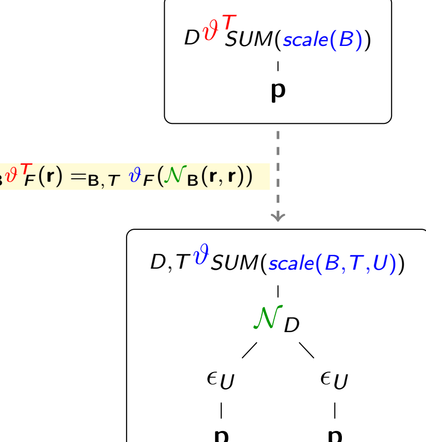
Temporal Op. = Primitive + Traditional Op.

Constructing Temporal Algebra Expressions

- **Query:** Temporal aggregation

$$_D\vartheta^T_{SUM(scale(B))}(\mathbf{p})$$

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



Mapping to SQL

- Our implementation provides direct access to primitive operators:

$\epsilon_U(\mathbf{r})$: `SELECT Ts Us, Te Ue, * FROM r`

$\mathcal{N}_\mathbf{B}(\mathbf{r}, \mathbf{s})$: `FROM (r NORMALIZE s USING(B) WITH (Ts, Te, Ts, Te)) r`

$\phi_\theta(\mathbf{r}, \mathbf{s})$: `FROM (r ALIGN s ON` θ `WITH (Ts, Te, Ts, Te)) r`

$\alpha(\mathbf{r})$: `SELECT ABSORB * FROM r`

- The source code of PostgreSQL with the complete temporal functionality integrated into the kernel of PostgreSQL at http://tpg.inf.unibz.it

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, Jul]
7	[May, Jul]

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

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

X	T
5	[Feb, Apr]
7	[May, Jul]

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, Jul]
7	A	[May, Jul]
3	B	[Apr, Aug]

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

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

X	Y	T
5	A	[Feb, Apr]
7	A	[May, Jul]
3	B	[Apr, Aug]

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 the largest X?

Input:

r

X	Y
5	A
7	A
3	B

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

```
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, Jul]
7	A	[May, Jul]
3	B	[Apr, Aug]

$\mathbf{r} \triangleright^{T}_{s.X>r.X} \mathbf{r}/\mathbf{s} = \phi_{\dots}(\mathbf{r},\mathbf{s}) \triangleright_{\dots \wedge \mathbf{r}.T=\mathbf{s}.T} \phi_{\dots}(\mathbf{s},\mathbf{r})$

```
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, Apr]
7	A	[May, Jul]
3	B	[Aug, Aug]

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
Tim	184K	CS
Joe	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, Jul]
Tim	184K	CS	[May, Jul]
Joe	153K	M	[Apr, Aug]

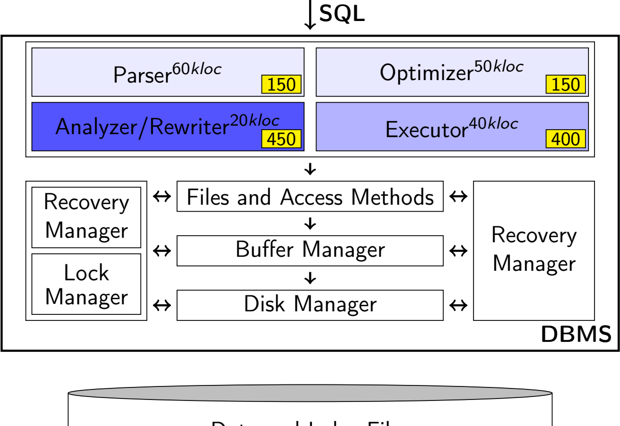
$_{D}\vartheta^{T}_{SUM(scale(B))}(\mathbf{p})$

```
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, Apr]
CS	276K	[May, Jul]
M	153K	[Apr, Aug]

PostgreSQL Implementation/1

- DBMS kernel integration of temporal primitives.



PostgreSQL Implementation/2

1. Modify Query Flow Trees

   SQL query
   $\xrightarrow{\text{parser}}$ parse tree
   $\xrightarrow{\text{analyzer}}$ query tree
   $\xrightarrow{\text{optimizer}}$ plan tree
   $\xrightarrow{\text{executor}}$ execution tree

2. Create Executor Functions

   `ExecInit`⟨`Operator`⟩
   `Exec`⟨`Operator`⟩
   `ExecEnd`⟨`Operator`⟩

PostgreSQL Implementation/3

- **Implementation Approach**
  - Temporal primitives are new nodes in query/plan/executor trees
  - Primitives themselves reuse traditional database operations
  - e.g., temporal alignment:
    1. *Join* matching tuples by DBMS internal left outer join
    2. *Sort*
    3. Apply *plane-sweep algorithm* to perform alignment
  - Only one new Executor function
- **Advantages**
  - Temporal primitives are optimized in the plan tree
    - Cost estimation
    - (Join) order
    - Selection push-down
    - Propagate orderings
  - Traditional database operations are optimized out of the box

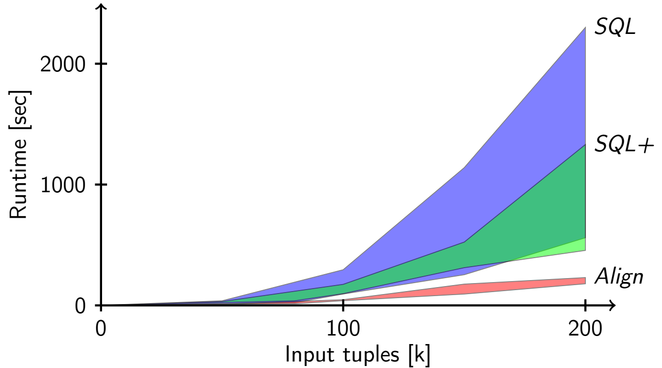
Empirical Evaluation

- Datasets
  - Real world dataset Incumben University of Arizona
  - Synthetic datasets
- Comparison for temporal outer joins

Align	Temporal Alignment and Reduction Rules
SQL	Plain SQL solution[1]
SQL+	SQL Join + $\mathcal{N}$ for the negative part

Outer Joins

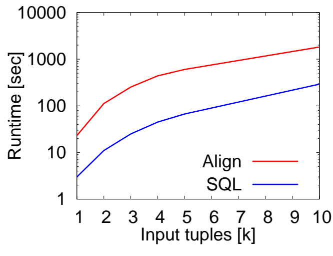
- Real world and random datasets
- Equi-Full and Theta-Left Outer Joins



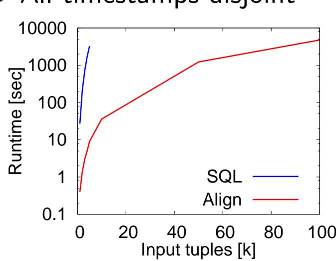
- SQL is inefficient and not robust for timestamp adjustment

Outer Joins SQL

- Left Outer Join ($\theta =$ `true`)
  - All timestamps equal



  - All timestamps disjoint



- *SQL* adjustment is based on `NOT EXISTS`
- *SQL* efficient when all timestamps are equal
  - Every tuple stops `NOT EXISTS`
- *SQL* inefficient when all timestamps are disjoint
  - All tuples need to be analyzed to stop `NOT EXISTS`

Lessons Learned

1. Integration into DBMS kernel adds a lot of precision and depth
2. Integration into DBMS kernel makes you understand DBMSs at a different level
3. "Going all the way" is really important for research
   - implement all examples
   - public source code
   - complete system
   - yes, the 80/20 rules applies :-)

Outline

1 Motivation

2 Background in Temporal Databases

3 History and State-of-the-Art

4 ANSI/ISO SQL:2011 Standard

5 SQL:2011 Standard in Commercial DBMSs

6 Native Support for Sequenced Temporal Data

7 **Summary and Outlook**

Summary

- Full-fledged temporal algebra/RDBMS for sequenced temporal queries (properties P1 + P2 + P3)
- Two primitives are required to adjust input relations
  - Temporal Normalizer
  - Temporal Aligner
- 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.
  - Leverage existing optimization and execution techniques.

Outlook

- SQL extension to express temporal queries
  - New keywords, statement level vs. operator level, . . .
- Physical query optimization
  - At the logical level two primitives are required to adjust periods.
  - Additional primitives can be used to boost performance
    - e.g., add a new primitive for full outer joins.
- Time series
  - Integrate time series data as first-class citizen into RDBMs
  - New operations: missing values, similarity search, ...
- Temporal relationships across different time points
  - Support for non-sequenced temporal semantics

Acknowledgments

- Joint work with
  - Michael Böhlen, University of Zurich
  - Anton Dignös, Free University of Bozen-Bolzano
  - Christian Jensen, Aalborg University
- Partially funded by the
  - Autonomous Province of Bozen-Bolzano
  - Swiss National Science Foundation
  - EU (Chorochronos)
  - FUB (TPG)

Thank You!