Database Technology for Processing Temporal Data

Michael H. Böhlen
University of Zurich, Switzerland
boehlen@ifi.uzh.ch
https://orcid.org/0000-0003-3694-9026

Anton Dignös
Free University of Bozen-Bolzano, Italy
dignoes@inf.unibz.it
https://orcid.org/0000-0002-7621-967X

Johann Gamper
Free University of Bozen-Bolzano, Italy
gamper@inf.unibz.it
https://orcid.org/0000-0002-7128-507X

Christian S. Jensen
Aalborg University, Denmark
csj@cs.aau.dk
https://orcid.org/0000-0002-9697-7670

Abstract

Despite the ubiquity of temporal data and considerable research on processing such data, database systems largely remain designed for processing the current state of some modeled reality. More recently, we have seen an increasing interest in processing historical or temporal data. The SQL:2011 standard introduced some temporal features, and commercial database management systems have started to offer temporal functionalities in a step-by-step manner. There has also been a proposal for a more fundamental and comprehensive solution for sequenced temporal queries, which allows a tight integration into relational database systems, thereby taking advantage of existing query optimization and evaluation technologies. New challenges for processing temporal data arise with multiple dimensions of time and the increasing amounts of data, including time series data that represent a special kind of temporal data.

2012 ACM Subject Classification Information systems → Data management systems, Information systems → Temporal data

Keywords and phrases Temporal databases, temporal query processing, sequenced semantics, SQL

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.2

Category Invited Paper

1 Introduction and Background

The storage and querying of temporal data in database management systems (DBMSs) has been researched for decades, evolving the field and covering multiple aspects of time, the design of SQL-based query languages, the development of efficient storage and index structures and algorithms, as well as standardization efforts. The last few years have seen a renewed interest in studying temporal data management.

To facilitate the formulation of temporal queries, various temporal query languages have been proposed [5]. The earliest proposals extended *SQL with new data types* with associated predicates and functions, e.g., as consolidated in TSQL2 [30]. Though simple, this approach makes it difficult to be comprehensive and to avoid unintended interactions among different temporal features. To overcome such problems, more systematic approaches were proposed that, conceptually, adopt a point-based view of data in combination with timestamp normalization to transform between an interval-based representation and the point-based conceptual model. Representative examples include IXSQL [22] that uses *fold* and *unfold* functions, SQL/TP [31] that uses a *normalization* function, and an approach [1] that extends normalization to bitemporal relations by means of a *split* operator. For a systematic construction of temporal SQL queries from nontemporal SQL queries, so-called *statement modifiers* were proposed in ATSQL [6].

To make the processing of temporal queries efficient, various query processing algorithms have been studied, primarily for temporal aggregations and temporal joins over interval timestamped relations. Different ways of grouping data along the time dimension yield different forms of temporal aggregation: instant, moving-window, and span temporal aggregation. Unified frameworks to express these forms of temporal aggregation were proposed [21, 4]. Prominent examples of index structures and algorithms for temporal aggregation include the *aggregation tree* algorithm [19] and the *balanced tree* algorithm [24] for instant temporal aggregation, the *SB-tree* [32], a disk-based index structure for the incremental maintenance of instant and moving-window temporal aggregates, and its extension, the *MVSB-tree* [34], which additionally supports nontemporal range predicates.

Joins are the second class of operators for which efficient evaluation algorithms have been studied intensively. An overview and classification of temporal join algorithms is available in the literature [13]. Recent research results include the *timeline index* [17, 18], a main memory index, which was further developed in the context of the *lazy endpoint-based interval* join algorithm [27]. The *overlap interval partition* join [10] partitions the input relations such that the percentage of matching tuples in corresponding partitions is maximized. This yields a robust algorithm that is not affected by the temporal distribution of the data. Another partition-based approach, the *disjoint interval partitioning* join algorithm [8], ensures that all tuples in a partition are temporally disjoint to avoid expensive backtracking. The forward-scan based *plane sweep* algorithm [7] tries to minimize the number of comparisons and provides also a parallel evaluation strategy based on a temporal partitioning of the two input relations.

Recently, we have seen a renewed interest in providing support for temporal data in database management systems in both academia and industry. This has several reasons: abundant storage has made long term archival of historical data feasible, and it has been recognized in many application areas that temporal data holds the potential to reveal valuable insights that cannot be found by analyzing only a snapshot of the data. This has been witnessed by the SQL:2011 standard [33, 20], which for the first time provides support for storing temporal data, and by commercial DBMSs that have started to offer temporal functionalities [26]. Recently, a comprehensive solution for sequenced queries has been integrated into the kernel of PostgreSQL [11].

2 Temporal Support in SQL:2011 and Commercial DBMSs

SQL:2011 Standard. The SQL:2011 standard [33, 20] is arguably the first SQL standard to introduce explicit support for the storage and manipulation of temporal data. A core extension concerns the possibility to specify one or two time periods associated with tables,

representing *application time* and *system time*, which are commonly known as valid time and transaction time, respectively [15]. Valid time is the time when a tuple is true in the modeled reality, whereas transaction time is the time when the tuple was current in the database. While the valid time is specified by users, the transaction time is maintained by the DBMS when a tuple is created, updated, or deleted.

SQL:2011 adopts an interval-based data model with tuple timestamping. Time periods can be added as metadata to the table schema, specifying a start time attribute and an end time attribute. As start and end time attributes are often already present, a time period can be added without modifying the table schema. This approach achieves backward compatibility, keeping old schemas, queries, and tools running.

The behavior of temporal tables in the case of updates and deletions is different for valid time tables and transaction time tables. Conventional update and delete operations on valid time tables work in the same way as for nontemporal tables. Additionally, tuples can be modified over parts of the associated time period, which might cause tuples to be split or cut. For transaction time tables, the user can only modify nontemporal attributes of the current tuples. The timestamp attributes are maintained automatically by the system whenever nontemporal attributes of current tuples are modified.

The SQL:2011 standard also specifies primary and foreign keys. A primary key on a valid time table can be used to ensure that only one value at a time exists for the nontemporal key attributes [16]. Foreign keys enforce the existence of certain tuples in a referenced table. Similarly, primary and foreign key constraints can be specified for transaction time tables.

The support for querying temporal relations is limited to simple range restrictions and predicates. For valid time tables, the usual SQL syntax can be used to specify constraints on the start and end time points of the periods. For transaction time tables, three new SQL extensions are provided to retrieve tuples in a given time range. There is no explicit support for more advanced operations, such as various forms of temporal aggregations or temporal joins.

Commercial DBMSs. Following the SQL:2011 standard, major database vendors have started to offer temporal support in their database management systems [26].

IBM offers the temporal features from SQL:2011 in version 10 of their DB2 database system [29], supporting both valid time and transaction time tables. Transaction time tables are implemented by means of a current table and a history table; queries over transaction time tables are automatically rewritten into queries over one or both of the two tables. The Oracle DBMS supports temporal features from SQL:2011 as of version 12c. The temporal features are implemented using the Oracle flashback technology [25] and adopt a syntax that differs slightly from the SQL standard. PostgreSQL version 9.2 introduces a new range data type together with associated predicates and functions into the language to support the SQL:2011 standard [28]. For efficient query processing over range predicates, two index structures have been provided, the Generalized Search Tree (GiST) [14] and the space-partitioned Generalized Search Tree (SP-GiST) [12]. The Teradata DBMS supports temporal features from the SQL:2011 standard from version 13.10 onwards [2]. For querying temporal tables, so-called temporal statement modifiers [6] are used in combination with query rewriting where temporal queries are translated into equivalent standard SQL queries. In terms of querying, Teradata is the most advanced database management system, supporting sequenced aggregation and coalescing. Since 2016, Microsoft's SQL Server [23] offers limited support for transaction time tables. For more general temporal query support, user-defined functions have to be used.

3 Native Support for Sequenced Temporal Queries

While the SQL:2011 standard provides limited support for querying temporal data, the *temporal alignment* framework [11] is the first approach to achieve systematic and comprehensive support for so-called sequenced temporal queries in relational database engines without limiting the use of queries with so-called nonsequenced semantics. The approach allows a tight integration into the database kernel, thus making it possible to leverage existing query optimization and evaluation strategies for processing temporal queries.

The key idea of the temporal alignment approach is to reduce temporal queries to nontemporal queries in a two-step process: (1) Adjust the timestamps of the input tuples such that they are aligned. This yields an intermediate relation, where all tuples that together contribute to a result tuple have the same timestamp. This intermediate relation can conceptually be considered as a sequence of snapshots, each of which lasts for one or more time points. Two interval adjustment operators are needed: a *temporal normalizer* for the operators π, ϑ, $-$, $\cap$, and $\cup$, and a *temporal aligner* for the operators $\times$, $\bowtie$, $⟕$, $⟖$, $⟗$, and $\triangleright$. (2) The corresponding nontemporal operator is applied to the intermediate relations. By treating the adjusted timestamps as nontemporal, atomic values and adding an equality constraint over the adjusted timestamps (e.g., as a grouping attribute for aggregation or an equality predicate in joins), it is ensured that tuples that contribute to the same result tuple are processed together, yielding the correct result of the original temporal query. Conceptually, the nontemporal query is applied on each snapshot of the intermediate relations.

The temporal alignment framework features two optional steps. First, it allows the replication of the original timestamp attribute as a nontemporal attribute before the interval adjustment step. This is necessary if a subsequent operation needs information about the original timestamp, e.g., a query predicate over the original timestamp. Second, it allows attribute values of a tuple to be "scaled" in response to changes to the duration of the tuple's timestamp, which may occur during the interval adjustment step.

The temporal alignment approach is systematic and separates interval adjustment from the evaluation of the operators. This strategy renders it possible to fully leverage the query optimization and evaluation engine of a DBMS for sequenced temporal query processing. An implementation of the temporal alignment framework in the kernel of the PostgreSQL database system is available at `tpg.inf.unibz.it` [9, 11].

4 Conclusion and Outlook

The processing of temporal data is receiving renewed attention in the database community. In this work, we provide a brief overview of current results, covering both research results and commercial database management systems (a more detailed version is available [3]). Starting with SQL:2011 the SQL standard offers support for storing and updating temporal data; query support remains limited. These temporal features have been implemented in a step-by-step fashion in prominent database management systems. In the research field, a number of new index structures and query algorithms, mainly for aggregation and join, have been proposed. Further, the first comprehensive framework for sequenced temporal queries has been implemented in a relational database management system.

Future work in temporal databases points in various directions. While temporal alignment provides a framework for implementing temporal query support in relational database systems, open issues remain that require further investigation. First, in order to achieve scalability to very large datasets in the framework, some operators need substantial performance improvements. This can be achieved, for example, by providing additional and more targeted

temporal alignment primitives that produce smaller intermediate relations. Also, as current cost estimates are very conservative, it is of interest to study more accurate and optimistic cost estimates for the query optimizer. This might require the maintenance of statistics about the temporal distribution of data in the dictionary. Integration of specialized query algorithms and equivalence rules may also be pertinent.

Second, it is of interest to broaden the applicability of the framework. For instance, it is of interest to extend the temporal alignment framework to relations with temporal duplicates. This extension is relevant since duplicates occur in many practical applications, and they are also permitted in the SQL:2011 standard. Another extension concerns the support for two or more time dimensions, such as valid time and transaction time. Currently, only valid time is supported, while the SQL:2011 standard supports both valid time and transaction time. One problem there is that the adjustment of bitemporal timestamps becomes much more complex. In time series data, a special type of temporal data, each value is timestamped with a time point rather than a time period. It is of interest to study how existing technologies from temporal databases can be adopted for the processing of such data.

Finally, more research in SQL-based temporal query languages is needed to facilitate the formulation of complex temporal queries; this aspect is not covered in the SQL:2011 standard.

References

1 Mikkel Agesen, Michael H. Böhlen, Lasse Poulsen, and Kristian Torp. A split operator for now-relative bitemporal databases. In *Proceedings of the 17th International Conference on Data Engineering, ICDE 2001*, pages 41–50, 2001. doi:10.1109/ICDE.2001.914812.

2 Mohammed Al-Kateb, Ahmad Ghazal, Alain Crolotte, Ramesh Bhashyam, Jaiprakash Chimanchode, and Sai Pavan Pakala. Temporal query processing in teradata. In *Proceedings of the 16th International Conference on Extending Database Technology, EDBT 2013*, pages 573–578, 2013. doi:10.1145/2452376.2452443.

3 Michael H. Böhlen, Anton Dignös, Johann Gamper, and Christian S. Jensen. Temporal data management - an overview. In Esteban Zimányi, editor, *Business Intelligence and Big Data - 7th European Summer School, eBISS 2017, Bruxelles, Belgium, July 2-7, 2017, Tutorial Lectures*, volume 324 of *Lecture Notes in Business Information Processing*, pages 51–83. Springer, 2017. doi:10.1007/978-3-319-96655-7_3.

4 Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Multi-dimensional aggregation for temporal data. In *Proceedings of the 10th International Conference on Extending Database Technology, EDBT 2006*, volume 3896 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2006. doi:10.1007/11687238_18.

5 Michael H. Böhlen and Christian S. Jensen. Temporal data model and query language concepts. In *Encyclopedia of Information Systems*, pages 437–453. Elsevier, 2003. doi:10.1016/B0-12-227240-4/00184-2.

6 Michael H. Böhlen, Christian S. Jensen, and Richard T. Snodgrass. Temporal statement modifiers. *ACM Trans. Database Syst.*, 25(4):407–456, 2000. URL: http://portal.acm.org/citation.cfm?id=377674.377665.

7 Panagiotis Bouros and Nikos Mamoulis. A forward scan based plane sweep algorithm for parallel interval joins. *PVLDB*, 10(11):1346–1357, 2017. URL: http://www.vldb.org/pvldb/vol10/p1346-bouros.pdf, doi:10.14778/3137628.3137644.

8 Francesco Cafagna and Michael H. Böhlen. Disjoint interval partitioning. *The VLDB J.*, 26(3):447–466, 2017. doi:10.1007/s00778-017-0456-7.

9 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012*, pages 433–444, 2012. doi:10.1145/2213836.2213886.

10 Anton Dignös, Michael H. Böhlen, and Johann Gamper. Overlap interval partition join. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2014*, pages 1459–1470, 2014. doi:10.1145/2588555.2612175.

11 Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational DBMS with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, 2016. doi:10.1145/2967608.

12 Mohamed Y. Eltabakh, Ramy Eltarras, and Walid G. Aref. Space-partitioning trees in postgresql: Realization and performance. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 100. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.146.

13 Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Join operations in temporal databases. *The VLDB J.*, 14(1):2–29, 2005. doi:10.1007/s00778-003-0111-3.

14 Joseph M. Hellerstein. Generalized search tree. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1222–1224. Springer US, 2009. doi:10.1007/978-0-387-39940-9_743.

15 Christian S. Jensen, Curtis E. Dyreson, Michael H. Böhlen, James Clifford, Ramez Elmasri, Shashi K. Gadia, Fabio Grandi, Patrick J. Hayes, Sushil Jajodia, Wolfgang Käfer, Nick Kline, Nikos A. Lorentzos, Yannis G. Mitsopoulos, Angelo Montanari, Daniel A. Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Michael D. Soo, Abdullah Uz Tansel, Paolo Tiberio, and Gio Wiederhold. The consensus glossary of temporal database concepts. In *Temporal Databases, Dagstuhl*, pages 367–405, 1997. doi:10.1007/BFb0053710.

16 Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng.*, 8(4):563–582, 1996. doi:10.1109/69.536250.

17 Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber, and Norman May. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pages 1173–1184, 2013. doi:10.1145/2463676.2465293.

18 Martin Kaufmann, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, and Franz Färber. Comprehensive and interactive temporal query processing with SAP HANA. *PVLDB*, 6(12):1210–1213, 2013. URL: http://www.vldb.org/pvldb/vol6/p1210-kaufmann.pdf, doi:10.14778/2536274.2536278.

19 Nick Kline and Richard T. Snodgrass. Computing temporal aggregates. In *Proceedings of the 11th International Conference on Data Engineering, ICDE 1995*, pages 222–231, 1995. doi:10.1109/ICDE.1995.380389.

20 Krishna G. Kulkarni and Jan-Eike Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3):34–43, 2012. doi:10.1145/2380776.2380786.

21 Inés Fernando Vega López, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal aggregate computation: a survey. *IEEE Trans. Knowl. Data Eng.*, 17(2):271–286, 2005. doi:10.1109/TKDE.2005.34.

22 Nikos A. Lorentzos and Yannis G. Mitsopoulos. SQL extension for interval data. *IEEE Trans. Knowl. Data Eng.*, 9(3):480–499, 1997. doi:10.1109/69.599935.

23 Microsoft. SQL Server 2016 - temporal tables. `https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables`, 2016.

24 Bongki Moon, Inés Fernando Vega López, and Vijaykumar Immanuel. Efficient algorithms for large-scale temporal aggregation. *IEEE Trans. Knowl. Data Eng.*, 15(3):744–759, 2003. `doi:10.1109/TKDE.2003.1198403`.

25 Oracle. Database development guide - temporal validity support. `https://docs.oracle.com/database/121/ADFNS/adfns_design.htm#ADFNS967`, 2016.

26 Dusan Petkovic. Temporal data in relational database systems: A comparison. In *New Advances in Information Systems and Technologies - Volume 1*, volume 444 of *Advances in Intelligent Systems and Computing*, pages 13–23. Springer, 2016. `doi:10.1007/978-3-319-31232-3_2`.

27 Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *Proceedings of the 32nd International Conference on Data Engineering, ICDE 2016*, pages 1098–1109, 2016. `doi:10.1109/ICDE.2016.7498316`.

28 PostgreSQL Global Development Group. Documentation manual PostgreSQL - range types. `http://www.postgresql.org/docs/9.2/static/rangetypes.html`, 2012.

29 Cynthia Saracco, Matthias Nicola, and Lenisha Gandhi. A matter of time: Temporal data management in DB2 10. `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf`, 2012.

30 Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

31 David Toman. Point vs. interval-based query languages for temporal databases. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1996*, pages 58–67, 1996. `doi:10.1145/237661.237676`.

32 Jun Yang and Jennifer Widom. Incremental computation and maintenance of temporal aggregates. *The VLDB J.*, 12(3):262–283, 2003. `doi:10.1007/s00778-003-0107-z`.

33 Fred Zemke. Whats new in SQL: 2011. *SIGMOD Record*, 41(1):67–73, 2012. `doi:10.1145/2206869.2206883`.

34 Donghui Zhang, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. Efficient computation of temporal aggregates with range predicates. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2001*, 2001. `doi:10.1145/375551.375600`.