Maintaining Bi-temporal Schema Versions in Temporal Data Warehouses

Anjana Gosain and Kriti Saroha

Abstract The temporal data warehouses (TDWs) have been proposed to correctly represent the revisions in dimension data. TDWs manage the evolution of schema and data with time through their versioning by time-stamping the data with valid time. Bi-temporal schema versioning in temporal data warehouses has also been discussed that not only allows retroactive and proactive schema revisions but also keeps track of them. The support for bi-temporal schema versioning raises an important issue for managing and storing the different versions of schema along with their data. The paper proposes an approach for managing bi-temporal schema versions in TDWs to allow for an effective management of several versions of schema along with their data.

Keywords Data warehouse · Temporal data warehouse · Schema versioning Transaction time · Valid time · Bi-temporal

1 Introduction

Data Warehouses (DWs) are a large reservoir of historical data, designed to provide support for multidimensional analysis and decision-making process [23, 25]. A DW schema as well as its data can undergo revisions with time to keep up with the application demands according to the user requirements and specifications. Several solutions have been presented in the literature to manage the revisions in DWs namely, schema and data evolution approach, schema versioning approach and temporal extensions. Schema and data evolution [5–7], [21, 22, 24, 28, 32] is a limited solution as it maintains only one DW schema and deletes the previous

A. Gosain (✉)
USICT, GGSIPU, Dwarka, India
e-mail: anjana_gosain@hotmail.com

K. Saroha
SOIT, CDAC, Noida, India
e-mail: kritisaroha@gmail.com

K. Saeed et al. (eds.), *Progress in Advanced Computing and Intelligent Engineering*,
Advances in Intelligent Systems and Computing 564,
https://doi.org/10.1007/978-981-10-6875-1_29

schema version incurring data loss. Schema versioning [2–4, 8, 9, 16, 27], on the other hand, preserves complete history of the DW evolution characterized by a set of schema revisions. But, it is also an established fact that not only the conceptual schema, but its underlying data may also evolve with time and thus, demand support for managing multiple versions of the schema as well as data. Temporal extensions [11, 13, 15, 26, 29], use time-stamps on the dimension data to fulfill this requirement and give rise to temporal data warehouse model with schema versioning support. TDWs use the research achievements of temporal databases and control the evolution of data with time-stamping of dimension data using their valid time. Valid time; determines the time for which an event is valid, and transaction time; indicates the time when an event is recorded in the database is generally used to keep a track and record of the revisions. At times, a combination of valid and transaction time (bi-temporal) may also be used [29].

Most of the works in the area of TDWs primarily deal with the evolution of dimension instances and use only the valid time for data and schema versioning. Schema and data versioning using valid time (Valid time versioning) is important for applications that require to handle retroactive (affecting things in the past) or proactive (what-if analysis) schema revisions but it fails to keep track of them (i.e., it only manages/implements the revision but does not keep track of when the revision was proposed). Bi-temporal versioning, on the other hand, not only manage but also keep track of retroactive and pro-active schema revisions, (i.e., keeps tracks of when a revision was proposed and when it was actually implemented) and has been discussed in the context of TDWs [18]. But, the paper does not discuss the storage options for the different bi-temporal versions of the schema as well as their data. This paper proposes an approach in the same direction.

In this paper, we aim to present bi-temporal schema versioning in TDWs with a wider aspect and discuss the storage options for several bi-temporal versions of schema and their data. The work extends the research achievements of temporal databases by time-stamping the schema versions with bi-temporal time [10, 19, 20, 30]. Two design solutions (central and distributed storage of data) are proposed to handle the extensional data when bi-temporal schema versioning is used. The support for bi-temporal schema versioning has been discussed at schema (intentional) along with data (extensional) level. Moreover, the concept of synchronous and non-synchronous alignment of data and schema is discussed in the context of TDWs (using valid time time-stamps on dimension data) and bi-temporal versioning of schema.

The rest of this paper is organized as follows. Section 2 presents a discussion on the related work. Section 3 gives an overview of different types of schema versioning and extensional data management. Section 4 presents the storage options for the management of bi-temporal versions of the schema with an example; and finally, in Sect. 5, we present the conclusions and final considerations.

2 Related Work

This section presents a discussion of the work done in the area of temporal data warehouses. Rasa et al. [1] recommended temporal star schema, using the valid time for time-stamping dimension and transaction data. Chamoni and Stock [11] proposed to store dimension structures with time-stamps containing valid times. Mendelzon and Vaisman [28] proposed a temporal design for multidimensional OLAP and also a temporal query language (TOLAP). The model apparently provided support for schema evolution by storing information related to the revisions in the instances and structure of the dimensions, but fail to record the history of data. A temporal multidimensional model was proposed by Body et al. [8, 9] that provides support to manage the evolution of multidimensional structures by using time-stamps on level instances along with their hierarchy and transaction data. However, only revisions to dimension schema and dimension instances have been discussed. Wrembel and Morzy [2] discussed Real and Alternate versions of multidimensional DWs but did not discuss the options to populate the new versions with data from previous versions. Golfarelli et al. [16] introduced the approach for schema augmentation but synchronous and non-synchronous mapping among data was not considered. The COMET model proposed by Eder and Koncilia [12–15] time stamps data with valid time to represent revisions in transaction and structure data. The model mainly deals with the evolution of dimension instances and does not include the evolution of schema or cubes. Mapping functions have also been proposed to allow transformations between structure versions using valid times but storage options have not been discussed for the versions. None of the approaches so far considered bi-temporal versioning of schema or data in TDWs.

3 Schema Versioning

Schema versioning done with respect to a single temporal dimension (transaction time or valid time) may be defined as transaction- or valid time versioning, respectively. Versioning that includes both temporal dimensions; transaction time as well as valid time produces bi-temporal versioning [14]. Thus, the versioning of schema can be categorized as transaction time, valid time or bi-temporal versioning.

- Transaction time Schema Versioning: Versioning of schema on transaction time, time-stamps all the versions of the schema with the related transaction time. It provides support only for on-time schema revisions, (i.e., revisions that are effective when applied) and that too, only in the current version of the schema. It does not allow for retro or proactive revisions.
- Valid time Schema Versioning: Schema versioning along valid time time-stamps all versions of schema with the associated valid time. The revised version of schema is effective only after its validity period is satisfied. In valid time schema versioning, multiple schema versions are accessible and any of the

schema versions can get affected by an update/revision if it either totally or partially overlaps with the valid time interval of the revision. It provides support for retro and proactive revisions but fails to keep a track of them.

- Bi-temporal Schema Versioning: This type of versioning time-stamps all the versions of schema using both transaction time and valid time. The transaction time is used to determine when the revision was suggested and the valid time indicates the duration for which the version of schema is valid. In bi-temporal schema versioning, only the present and the overlapped bi-temporal versions of schema can get affected by a schema revision. For a system that requires complete tractability, only bi-temporal schema versioning can establish that a new version of schema was generated as a result of a retro- or a pro-active schema revision [10].

Here, we propose design solutions for bi-temporal schema versioning in TDWs.

3.1 Design Choices for Handling Extensional Data

In temporal databases, two different storage solutions are discussed for managing extensional data namely, single pool and multi-pool solution [10]. We proposed to extend the concept for TDWs and presented two storage solutions [17]. The storage solutions and their response with respect to the bi-temporal schema versioning are discussed in this paper with the help of examples. The solutions are presented at the logical level, without moving into the physical design details.

Central Storage of Data, where only one data repository stores all data related to different versions of schema according to an extended schema, which contains all the attributes ever stated by any of the schema revisions.

Distributed Storage of Data, where multiple data repositories store data for various versions of schema. Each of the data repositories is configured corresponding to its related version of schema. To initialize a new storage, the records from the older storage are moved into the new storage according to the schema revisions.

In cases where both data and schema versioning are performed along the same temporal dimensions, any of the mappings; synchronous or non-synchronous may be used.

While using synchronous mapping, the version of schema having the same validity of records with reference to the common temporal dimension are used to record, extract and update data [10].

But in case of non-synchronous mapping, any of the versions may be opted to extract and update data irrespective of the valid time interval of the version of schema. Here, the validity of schema and data are independent even with regard to the common temporal dimension(s) [10].

Central storage of data is invariably synchronous, while distributed storage may be synchronous or non-synchronous.

4 Proposed Approach

Bi-temporal schema versioning preserves all the valid time versions of schema formed due to subsequent schema revisions. In bi-temporal schema versioning, only the current bi-temporal version of the schema that overlaps the validity interval specified for the schema update would be affected by the revisions. The user is allowed to select the bi-temporal schema versions to be included for modifications by specifying the validity of the schema revision. Moreover, bi-temporal schema versioning provides support for both implementing as well as tracking retro- and proactive schema alterations. The operations on bi-temporal schema versions and their data are described by means of examples and figures.

4.1 Managing Intensional Data

In bi-temporal schema versioning, a new version of schema is generated by implementing the required revisions to the present bi-temporal schema version that qualifies the specified valid time interval. The new bi-temporal schema version is assigned the validity period of the schema revision and current transaction time (NOW). Further, any of the older versions of schema that completely overlaps with the validity period of the revisions coexist with the new schema version but both have different transaction times. However, the validity interval of the older versions of schema, which have only partial overlap with the valid time interval of the revision applied is limited accordingly.

It is explained using an example as shown in Fig. 1. Suppose, an attribute p4 is removed from the bi-temporal schema version (SV) with validity [t′, t″]. The states of bi-temporal schema versions before the modification and following the modification are presented in Fig. 1a and b, respectively. Of all the bi-temporal versions of schema (SV_1 to SV_4) in the example, only two of them (SV_2 and SV_4) are partly overlapped and one (SV_3) totally overlaps with the valid time interval [t′, t″].

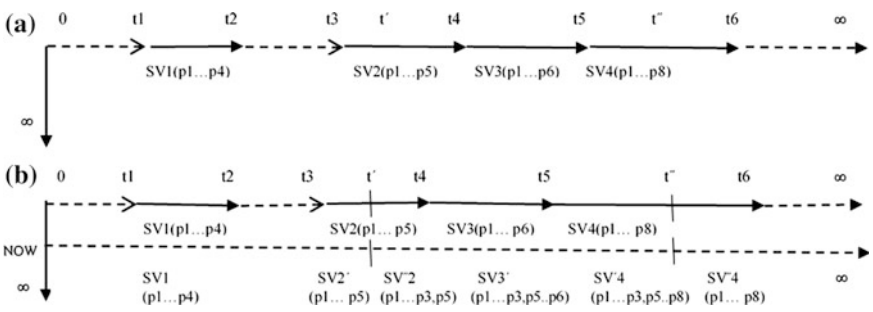

Fig. 1 Bitemporal versions of schema in temporal DW

The version of schema associated with [t_1, t_2] remains unaffected by the modification as it does not satisfy the valid time interval [t′, t″]. The schema versions associated with [t_3, t_4] and [t_5, t_6], partially overlaps the valid time interval [t′, t″] get affected by the modification and are thus split into two parts (SV′2, SV″2) and (SV′4, SV″4), respectively. The non-overlapping portion (SV′2, SV″4) is not affected by the modification and would retain all of its old attributes, whereas the overlapping portion (SV″2, SV′4) would remove the attribute p4 and results in (p1..p3, p5) and (p1..p3, p5..p8), respectively. The bi-temporal schema version related to [t_4, t_5] is also affected by the modification as it totally overlaps the valid time interval [t′, t″]. Therefore, p4 is removed from the new version of schema and creates SV′3 with attributes (p1..p3, p5..p6).

4.2 Managing Extensional Data

Since the TDWs mainly contains the valid time, bi-temporal schema versioning would be synchronous along transaction time and either synchronous or non-synchronous along valid time.

4.2.1 Central Storage of Data (Synchronous and Non-synchronous)

The central storage solution maintains only one data repository to store the complete data according to an extended schema version that contains all the attributes ever introduced by subsequent schema revisions [10]. The data repository can only grow, i.e., no attribute or temporal dimension is ever dropped from the data repository as a result of a schema revision and the revision can only be recorded in the meta-schema. But, if a schema revision results in the addition of an attribute, or a temporal dimension, the complete data repository is updated to the revised form. Data are thus stored using the enlarged schema format defined by successive schema revisions and the history of the revisions are recorded only in the meta-schema.

The information stored in meta-schema helps to restore the initial or previous structures of data for any of the bi-temporal versions of schema, if required.

4.2.2 Distributed Storage of Data

The distributed storage solution maintains different data repositories/stores for different versions of schema formed in response to schema revisions. A new data store is constructed according to the schema revisions and each version of the schema is allowed to access only its own data store. The new data store is populated with only the current data records from the older store associated with the modified version of schema and updated according to the revisions applied. The valid time

intervals of the records remain unaffected in the case of non-synchronous mapping, and their time-stamps are confined in the case of synchronous mapping.

Distributed Storage of Data (Synchronous).

Synchronous mapping of the distributed storage of data restricts the validity interval of data records in all the new data stores according to their intersection with the validity period of the version of schema. Also, for the management of data stores associated with versions of schema that have partial overlap with the revisions applied to the schema, the valid time interval of the records must be limited according to the valid time interval of their corresponding schema version.

Figure 2 shows the result of a modification (adding an attribute p3) which overlaps SV1 on [t2, t3], for synchronous mapping. It results in a new schema version SV2 that consist of attributes p1, p2, and p3. Figure 2a represents the initial data store SV1 with valid time interval [t1, t3]. The temporal pertinence and data records of distributed data stores are given in Fig. 2c. In the distributed storage solution, a new data store has to be created to support the new schema version SV2. The initial records are segmented in accordance with their valid time intervals and are subsequently divided between the two data stores. In the central storage solution, the records of the data store are not partitioned as shown in Fig. 2b.

It may be noted that the above constraint might result in loss of details about the original valid time interval of extensional data. It is evident from the example that for some records (e.g., (A1, B1)), the data may have duplicate copies in both newly created versions, where the validity period gets divided corresponding to the synchronous management of data.

Distributed Storage of Data (Non-synchronous).

Using non-synchronous distributed storage solution, the format of all data stores is according to the associated version of schema. Thus, a new data store is constructed and it is loaded with data by copying the data from the initial data store affected by the modifications specified for the older schema (update/drop of an attribute etc.).

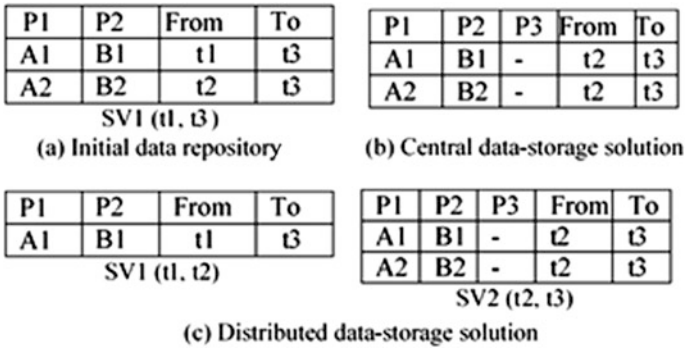

Fig. 2 Synchronous central and distributed storage solution

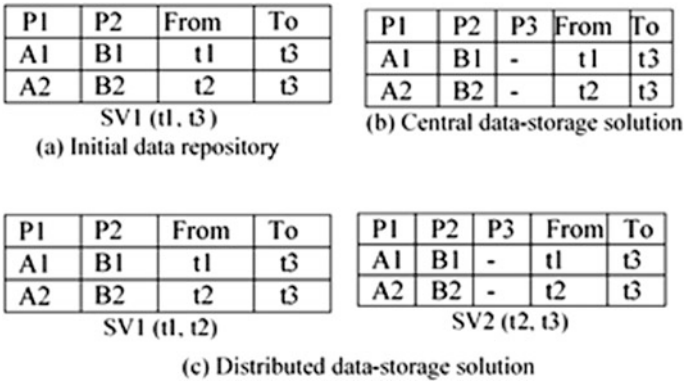

Fig. 3 Non-synchronous central and distributed storage solution

An example for non-synchronous mapping of extensional data is shown in Fig. 3. If the same schema modification is used (as given in Fig. 2) for the non-synchronous mapping, it does not require to partition the valid time interval for distributed storage of data. For this example, the results of central storage match with the new data store generated for distributed storage of data and it is notable that the data is not duplicated in this case.

5 Conclusion

The approach proposed in the paper would allow to trace the history of schema with bi-temporal versioning, assures consistency of data and presents different options to optimize the usage of space for storage of data. Bi-temporal versioning of schema and valid time versioning of data, together with the data storage solutions and the correlation between schema and data versioning, were analyzed to explore the possibilities to model a temporal data warehouse that supports bi-temporal schema versioning. The choice for the storage option is dependent on the availability of storage space; for example, if the available space is restricted then central storage of data may be selected. The preference for synchronous and non-synchronous management depends on the degree of freedom required among the data elements. The central storage solution does not create duplicate copies of data, but might require more storage space as it enlarges the format of the schema after revisions. On the other hand, the distributed storage duplicate data from the affected pool(s).

The distributed storage solution has an after-effect on the data model as the various data stores would support independent evolutions of the data by applying modifications through different versions of the schema. For some applications that need data from both older and newly created schema versions, it may be required to rebuild the entire history of schema versions if synchronously distributed storage is

used. This is because the records in the data stores are divided in synchronism with the distinct versions of the schema. On the other hand, in case of non-synchronous mapping, the full history of data is maintained for every version of the schema. Therefore it allows old applications to execute properly on older data as before, although newer applications are required to be developed for new data that contains the new attributes. Also, with synchronously distributed storage, data cannot be queried using the older details of schema because the data is updated only in the current data store.

Furthermore, if queries span over multiple schemas, when central storage solution is employed, then the solution generated for the query would comprise of only a single table. But, if distributed storage is adopted, then a new version of schema is required to be created that includes all the attributes needed for the solution of the query. The work can be extended to support bi-temporal schema versions in bi-temporal data warehouses.

References

1. Agrawal, R., Gupta, A., Sarawagi, S.: Modeling multidimensional databases. IBM Research Report, IBM Almaden Research Center (1995)
2. Bębel, B., Eder, J., Konicilia, C., Morzy, T., Wrembel, R.: Creation and management of versions in multiversion data warehouse. In: Proceedings of ACM Symposium on Applied Computing (SAC), pp. 717–723 (2004)
3. Bębel, B., Królikowski, Z., Wrembel, R.: Managing multiple real and simulation business scenarios by means of a multiversion data warehouse. In: Proceedings of International Conference on Business Information Systems (BIS). Lecture Notes in Informatics, pp. 102–113 (2006)
4. Bębel, B., Wrembel, R., Czejdo, B.: Storage structures for sharing data in multi-version data warehouse. In: Proceedings of Baltic Conference on Databases and Information Systems, pp. 218–231 (2004)
5. Benítez-Guerrero, E., Collet, C., Adiba, M.: The WHES approach to data warehouse evolution. Digit. J. e-Gnosis (2003). http://www.e-gnosis.udg.mx, ISSN No. 1665–5745
6. Blaschka, M., Sapia, C., Hofling, G.: On schema evolution in multidimensional databases. In: Proceedings of International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Lecture Notes in Computer Science, vol. 1676, pp. 153–164 (1999)
7. Blaschka, M.: FIESTA: A framework for schema evolution in multidimensional information systems. In: 6th CAiSE Doctoral Consortium. Heidelberg (1999)
8. Body, M., Miquel, M., Bédard, Y., Tchounikine, A.: A multidimensional and multiversion structure for OLAP applications. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP), pp. 1–6 (2002)
9. Body, M., Miquel, M., Bédard, Y., Tchounikine, A.: Handling evolutions in multidimensional structures. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 581 (2003)
10. De Castro, C., Grandi, F., Scalas, M., R.: On Schema Versioning in Temporal Databases. In: Clifford, S., Tuzhilin, A. (eds.) Recent Advances in Temporal Databases, pp. 272–294. Springer, Zurich Switzerland (1995)
11. Chamoni, P., Stock, S.: Temporal structures in data warehousing. In: Proceedings of International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Lecture Notes in Computer Science, vol. 1676, pp. 353–358 (1997)

12. Eder, J.: Evolution of dimension data in temporal data warehouses. Technical Report 11, Univ. of Klagenfurt, Dep. of Informatics-Systems (2000)
13. Eder, J., Koncilia, C.: Changes of dimension data in temporal data warehouses. In: Proceedings of International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Lecture Notes in Computer Science, vol. 2114, pp. 284–293 (2001)
14. Eder, J., Koncilia, C., Morzy, T.: A model for a temporal data warehouse. In: Proceedings of the International OESSEO Conference. Rome Italy (2001)
15. Eder, J., Koncilia, C., Morzy, T.: The COMET metamodel for temporal data warehouses. In: Proceedings of Conference on Advanced Information Systems Engineering (CAiSE). Lecture Notes in Computer Science, vol. 2348, pp. 83–99 (2002)
16. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses. In: Proceedings of ER Workshops. Lecture Notes in Computer Science, vol. 3289, pp. 415–428 (2004)
17. Gosain, A., Saroha, K.: Storage structure for handling schema versions in temporal data warehouses. In: Accepted in 4th International Conference on Advanced Computing, Networking, and Informatics (ICACNI) (2016)
18. Gosain, A., Saroha, K.: Bi-temporal schema versioning in temporal data warehouses. In: Communicated in International Conference on Frontiers of Intelligent Computing: Theory and applications (FICTA) (2016)
19. Grandi, F., Mandreoli, F., Scalas, M.: A generalized modeling framework for schema versioning support. In: Australasian Database Conference, pp. 33–40 (2000)
20. Grandi, F., Mandreoli, F., Scalas, M.: A formal model for temporal schema versioning in object oriented databases. Technical report CSITE-014–98, CSITE-CNR (1998)
21. Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A.: Maintaining data cubes under dimension updates. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 346–355 (1999)
22. Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A.: Updating OLAP dimensions. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP), pp. 60–66 (1999)
23. Inmon, W.H.: Building the Data Warehouse. Wiley (1996)
24. Kaas, C.K., Pedersen, T.B., Rasmussen, B.D.: Schema evolution for stars and snowflakes. In: Proceedings of International Conference on Enterprise Information Systems (ICEIS), pp. 425–433 (2004)
25. Kimball, R., Ross, M.: The Data Warehouse Toolkit. Wiley (2002)
26. Letz, C., Henn, E.T., Vossen, G.: Consistency in data warehouse dimensions. In: Proceedings of International Database Engineering and Applications Symposium (IDEAS), pp. 224–232 (2002)
27. Malinowski, E., Zimanyi, E.: A conceptual solution for representing time in data warehouse dimensions. In: 3rd Asia-Pacific Conference on Conceptual Modelling, Hobart Australia, pp. 45–54 (2006)
28. Mendelzon, A.O., Vaisman, A.A.: Temporal queries in OLAP. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 242–253 (2000)
29. Schlesinger, L., Bauer, A., Lehner, W., Ediberidze, G., Gutzman, M.: Efficiently synchronizing multidimensional schema data. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP), pp. 69–76 (2001)
30. Serna-Encinas, M.-T., Adiba, M.: Exploiting bitemporal schema versions for managing an historical medical data warehouse: a case study. In: Proceedings of the 6th Mexican International Conference on Computer Science (ENC'05), pp. 88–95. IEEE Computer Society (2005)
31 Vaisman, A., Mendelzon, A.: A temporal query language for OLAP: implementation and case study. In: Proceedings of Workshop on Data Bases and Programming Languages (DBPL). Lecture Notes in Computer Science, vol. 2397, pp. 78–96. Springer (2001)