Querying Multiversion Data Warehouses

Waqas Ahmed[1,2](✉) and Esteban Zimányi[1]

[1] Department of Computer & Decision Engineering (CoDE),
Université libre de Bruxelles, Brussels, Belgium
{waqas.ahmed,ezimanyi}@ulb.ac.be

[2] Institute of Computing Science, Poznan University of Technology, Poznan, Poland

Abstract. Data warehouses (DWs) change in their content and structure due to changes in the feeding sources, business requirements, the modeled reality, and legislation, to name a few. Keeping the history of changes in the content and structure of a DW enables the user to analyze the state of the business world retrospectively or prospectively. Multiversion data warehouses (MVDWs) keep the history of content and structure changes by creating multiple data warehouse versions. Querying such DWs is complex as data is stored in multiple schema versions. In this paper, we discuss various schema changes in a multidimensional model, and elaborate their impact on the queries. Further, we also propose a system to support querying MVDWs.

1 Introduction

Data warehouses (DWs) store historical, subject-oriented, and often heterogeneous data that is fed by external data sources (EDSs). An inherent characteristic of these data sources is that they change in their content and structure independently of the DW that integrates data from them.

Data warehouses are modeled as multidimensional (MD) cubes and analytical applications query these cubes to produce reports. A MD cube consists of facts, measures and dimensions. A *fact* is a focus of analysis and a *measure* is an associated numerical value that quantifies the fact. For example, the analysis of sales in a store is a fact and the quantity of items sold can be a measure to analyze the sales. *Dimensions* provide various perspectives to analyze a fact. For example, a dimension customer can be used to analyze the sales made by a store to a particular age group of customers. Dimensions consist of discrete alphanumeric attributes which are organized in *hierarchies.* Each set of distinct attributes of a dimension hierarchy is called a *level.* Hierarchies allow decision-makers to analyze measures at various levels of detail. An example of a hierarchy that belongs to dimension geography is store→city→region where store, city, and region are the levels of hierarchy and each level stores specific characteristics about the dimension. Instances of a level are called *members.*

W. Ahmed—This research is partially funded by the Erasmus Mundus Joint Doctorate IT4BI-DC

DOI: 10.1007/978-3-319-23201-0_36

The MD model was based on the assumption that the content and structure of a DW remain fixed, but the practice has proved this assumption wrong. The content and structure of a MD cube may change due to changes in feeding sources, the business requirements, the modeled reality, the analysis requirements, and legislation, to name a few. Maintaining the history of changes in the content and structure of a DW enables users to analyze the state of the business world in the past or future, and in some cases, it may be required for audit and accountability purposes. A change in the value of dimension attribute is an example of content change, whereas a change in the structure of a hierarchy is an example of schema change. Three alternative approaches, namely slowly changing dimensions (SCDs) [10], temporal data warehouses (TDWs) [2,11], and multiversion data warehouses (MVDWs) [17,18] have been proposed to address the issue of content and schema changes in DWs. SCDs only handle changes in dimension members. TDWs provide support for storing and querying time-varying dimension members and facts. MVDWs maintain the history of content and schema changes by creating multiple DW versions.

Answering queries that require data from multiple schema versions (called cross-version queries) is not trivial, in particular because of the data in multiple versions may have different structure or the data may be present in one version but missing in another. To address the issue of querying MVDWs, in this paper (1) we provide a detailed discussion about schema changes in the MD model and their impact on user queries, and (2) based on our discussion we propose a system to support querying MVDWs.

The rest of this paper is organized as follows. Section 2 reviews related work. In Sect. 3, we introduce a running example that will be used throughout the rest of the paper. In Sect. 4, we discuss schema changes in the MD model and their impact on user queries. In Sect. 5, we provide three possible approaches to manage schema changes in relational databases, highlight the differences among queries for each approach, and propose a system to support querying MVDWs. Finally, Sect. 6 concludes the paper and provides considerations for future research.

2 Related Work

The challenge of managing schema changes in database systems and querying data across multiple schema versions is not new and the issues related to the problems are discussed in [14]. In the literature, the solutions to managing schema changes in databases are classified into three broad categories [6,14]. *Schema modification* allows changes in the schema but as a result, the existing data may become unavailable. *Schema evolution* supports schema changes while preserving existing data. Finally, *schema versioning* enables to store data in multiple schema versions.

In [3] is presented a system to automate database schema and integrity constraint evolution. The presented PRISM and PRISM++ systems automatically rewrite queries to support legacy applications and data migration. The

PRIMA [13] system also provides a mechanism to archive and query historical data. In [8], the authors examined data and schema evolution in a branched environment where data can evolve simultaneously under various schema versions. The schema definitions are stored as XML-based documents and the data records are stored in relational columns. In all of the these approaches, the challenge of querying data from multiple schema versions is not addressed.

Golfarelli et al. [5] presented an approach that uses a graph-based metaschema to create and query multiple schema versions in a DW. They introduced the concept of an augmented schema to handle the issue of missing data between versions. When the user creates a new schema version from an existing one, an augmented schema is also associated with the old version: It is the most generic schema containing all the elements from both the new and the old versions. In [19], the authors also presented a metadata-based version management system for MVDWs. In both of the above mentioned approaches, to answer a cross-version query, firstly the user query is converted into individual queries against each version, and then the results of these individual queries are combined and presented to the user.

The model presented in [4] supports changes in the structure of dimension members and also provides a list of integrity constraints to maintain the consistency of data across multiple versions. Although the authors mentioned about querying multiple schema versions, the question of how to answer cross-version queries remained unanswered.

3 Running Example

Multiversion DWs manage the evolution of their content and structure by creating multiple DW versions. Each DW version consists of a schema version and an instance version. The *schema version* defines the structure of data during a specific period, whereas the *instance version* consists of the data stored using a particular schema version. At a given instant, only one DW version is used to store data and it is called the *current version*. Although it is possible to derive multiple schema versions from the current version, for the sake of simplicity, we only consider the sequential versioning approach [19], in which a new version can be derived by applying changes to the current version only.

Each version has an associated begin application time (BAT) and end application time (EAT) that represent a close-open interval during which a version is used to store data. The interval [BAT, EAT) is called the *validity period* of the version. The EAT of the current version is set to a special value "*UC*" (until changed). A more detailed and formal definition of a MVDW can be found in [1]. Figure 1 shows the multiple versions in an example MVDW (for the moment, ignore the query types in the figure).

We will use the following motivating example throughout the rest of the paper. Our example MVDW is used to analyze the sales of a company. The cube can be modeled as either a star or snowflake schema. In our example, we consider a snowflake schema as it is more difficult to manage its evolution as compared

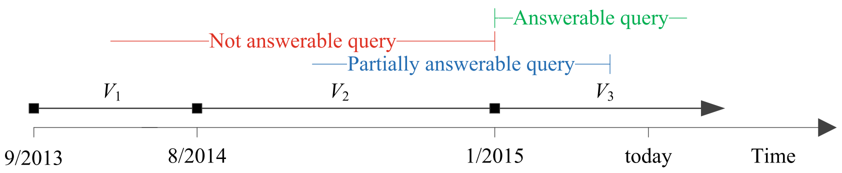
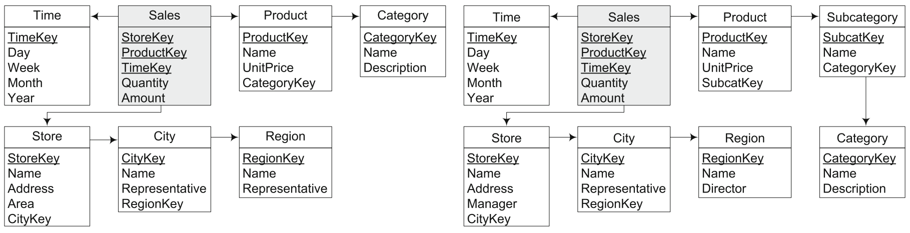
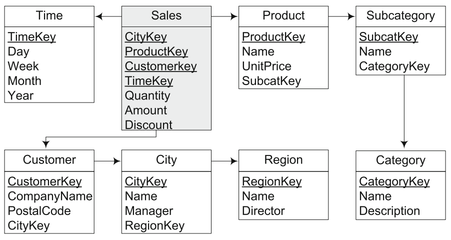


Fig. 1. Three versions of a DW and possible cases for a query computing the value of a schema element present in the current version only

to a star schema [9]. The initial version $V_1[9/2013, UC)$ of the DW was created in September 2013 and consisted of fact Sales and dimensions Time, Geography, and Product. Dimension Geography consisted of levels Store, City, and Region. Dimension Product consisted of levels Product and Category. Figure 2a shows the schema of the DW in version V_1.



(a) DW schema in version V_1 **(b)** DW schema in version V_2



(c) DW schema in version V_3

Fig. 2. Schema versions in the example MVDW

In August 2014, the user extended the Product dimension by inserting a new level Subcategory between Product and Category. Also, attribute Area was removed and attribute Manager was added to level Store. These changes resulted version $V_2[8/2014, UC)$. As a consequence, the validity period of V_1 was changed to $[9/2013, 8/2014)$. Figure 2b shows the schema of the DW in version V_2. In January 2015, the user deleted level Store from Geography and made City the base level of the dimension. She also added a new dimension Customer and a measure Discount to Sales. Furthermore, attribute Representative of level City was renamed to Director. As a result, version $V_3[1/2015, UC)$ was derived from V_2. Figure 2c shows the DW schema in version V_3.

4 Schema Changes in a Data Warehouse

The possible changes to a dimension schema include: (1) adding a new level to a dimension, (2) deleting a level from a dimension, (3) adding an attribute to a level, (4) removing an attribute from a level, (5) renaming an attribute of a level, and (6) changing the domain of an attribute of a level. Similarly, the possible changes to a fact schema include: (1) adding a new dimension to the fact, (2) deleting a dimension from the fact, (3) adding a new measure to the fact, (4) deleting a measure from the fact, (5) renaming a measure in the fact and, (6) changing the domain of a measure in the fact. We discuss next the impact of these changes on querying a MVDW. For the discussion, we will use the queries specified in Table 1.

Table 1. Queries implying data from multiple schema versions

No.	Query	Answerable
Q1	Total sales amount per customer and month from December 2013 till August 2014.	No
Q2	Total sales amount per customer from January 2015 until today.	Yes
Q3	Total sales amount per customer and month from June 2014 till June 2015.	Partially
Q4	Total sales amount per category until today.	Yes
Q5	Total sales amount per subcategory for December 2014.	No
Q6	Total sales amount per subcategory until today.	Partially
Q7	Total quantity sold per product and city until today	Yes
Q8	Total quantity sold per product and store in April 2015	No

Adding and/or Removing Levels. The addition of a new base level to a dimension increases the granularity at which the facts are stored for that dimension. The effect of this addition is similar to adding a new dimension to a cube. Consider the addition of dimension Customer in version V_3. This change requires not only adding a new dimension but also a new version of the fact so that all newly added facts may have a customer dimension member associated with them. It is worth noticing that the customer information related to sales will not be available for the facts stored before January 2015 because the earlier versions did not include the customer dimension. Query Q1 cannot be answered because dimension Customer did not exist between December 2013 and August 2014 and the sales facts stored for this period do not have any customer members associated with them. Query Q2 can be answered without any problem because sales facts from January 2013 until today have associated customer members. However, query Q3 can be answered partially, only from January 2015 onwards.

The addition of a new level to a dimension other than at the base level does not require the creation of a new fact version. Consider the schema change where level Subcategory was added between Product and Category. We assume that the user does not assign the existing products to the subcategories and only newly added products are assigned to the subcategories. Now consider query Q4, which computes the total sales amount per category until today. Since the product to category relation exists in all the versions of DW, the query can be answered and the result set can be computed by traversing through two different lattices, i.e., from September 2013 till July 2014 this information is directly available, whereas from August 2014 until today, the products' categories can be obtained from the subcategories.

The effect of deleting a base level is almost similar to adding a new dimension because a new base level of the dimension has to be defined. In our example, in version V_3, level Store was deleted from dimension Geography and City was made the new base level. Consider query Q7, which requires to compute the total quantity of a given product sold per city. The query can be answered completely because in earlier versions V_1 and V_2, measure Quantity in fact Sales can be related to the cities through level Store, whereas in the current version this information is directly available. In case of query Q8, level Store ceased to exist after January 2015, hence it cannot be answered.

Adding and/or Removing Attributes. The schema of dimensions change by adding or removing attributes in a level. For example, in version V_2, attribute Area is deleted from level Store and Manager is added to the same level. The value of attribute Area will not be available for store members that were stored after August 2014. Similarly, the information about the managers of stores, which existed in V_1, will not be available.

Other changes in the attributes such as, changes in the domain of an attribute, or renaming an attribute are trivial and can be handled either by modifying the existing attribute or adding a new attribute for each change. However, in case of renaming an attribute, it is important to keep track of attribute names in each version so that cross-version queries could be supported. In V_2, attribute Representative of level City was renamed to Director and it is important to keep the record of this change to support the queries asking for Representative and/or Director during the validity period of any of these two versions.

Adding or Removing Dimensions. The addition or removal of a dimension from a fact changes the schema of the fact. It is similar to adding or removing a base level of a dimension. The effect of changes in the measures is similar to changes in the attributes of dimensions and such changes can be managed in the same manner as in case of changes in the attributes.

To sum up our discussion in this section, consider a query Q that computes the value of a schema element E over a period P, such that E is present in the current version only. There are three possibilities for Q based on P: (1) if P is contained within the validity period of the current version then Q is answerable, (2) if P overlaps with the validity period of current version, Q is

Store Key	Product Key	Time Key	Quantity	Amount
s1	p1	t1	5	20
s2	p1	t1	3	15
s3	p2	t2	2	18
s4	p2	t3	3	9

(a) Sales in V_2 using the STV approach

Store Key	City Key	Product Key	Time Key	Cust. Key	Quantity	Amount	Discount
s1	null	p1	t1	null	5	20	null
s2	null	p1	t1	null	3	15	null
s3	null	p2	t2	null	2	18	null
s4	null	p2	t3	null	3	9	null
null	c1	p3	t4	cu1	6	8	0.10

(b) Sales in V_3 using the STV approach

City Key	Product Key	Time Key	Cust. Key	Quantity	Amount	Discount
c1	p3	t4	cu1	6	8	0.10

(c) Sales in V_3 using the MTV approach

Fig. 3. State of the Sales table in various versions using the STV and MTV approaches

partially answerable, and (3) otherwise, it is not possible to answer Q because the value of E is unavailable. Figure 1 shows these three cases.

5 Manipulating Multiversion Data Warehouses

Three approaches [1,16] have been proposed for storing multiple versions of a data warehouse, namely, (1) single-table version (STV), (2) multiple-table version (MTV), and (3) hybrid-table version (HTV). In this section, we briefly discuss each of these approaches and their impact on multiversion queries.

In the STV approach, each table has only one version throughout the lifetime of the DW. Each table contains all attributes that have ever been defined for it. This means that it is an append-only approach and the new attributes are added to the table. In this approach, the schema of the DW is always growing. For every new record, a default or null value is stored for the deleted attributes. Figures 3a and 3b show fact Sales in versions V_2 and V_3, respectively. Notice that new attributes CityKey and CustomerKey are added to link new levels City and Customer. Similarly, a new measure Discount is added. The sales records stored using earlier versions were not linked to the customers and cities, therefore for those records null values are stored in these attributes. Although level Store is deleted in the new version, the corresponding attribute StoreKey in Sales is not deleted and all new records store a null value for it.

In the MTV approach, each change in the schema of a table produces a new version of the table and new data is stored using this new version. Contrary to the STV approach, attributes can be added or removed in the new version. Figure 3c shows the Sales in V_3 using the MTV approach. Notice that the deleted attribute StoreKey does not exist in the new version. The MTV approach avoids the space overhead which is incurred in case of the STV approach but in some scenarios, it may require creating of fact versions because of new dimension versions. For example, if a new version of a table corresponding to the base level of a dimension is created, this requires to create a new fact version as well. Furthermore, all valid dimension members from the previous dimension version must be inserted into the new version so that they could be linked to the incoming facts, if needed.

The HTV approach combines the advantages of the STV and the MTV approaches. It creates a new version only in case of the schema changes in facts but maintains a single version for dimension tables. In this way, the problem of space overhead and complexity of managing multiple dimension versions and members loading can be avoided. In our example, in version V_3, fact Sales consists of two tables shown in Fig. 3c and 3a, respectively. However, the Product and Store tables will consist of a single version as in the STV approach.

Analytical queries are complex in nature since they often involve aggregation of data and joins across multiple dimensions. Querying a MVDW is even more complex since the data may be stored across multiple versions with different structure. Furthermore, as discussed above, queries to a MVDW also depend upon the approach used for storing the various versions. For example, if the user of our example MVDW is interested in average yearly sales for each city, then for the STV approach, it can be computed by query shown in Fig. 4a. The same query is different for the MTV approach because as a result of the first schema change, a new version of Store was created. To link this new version with the fact, a new version of the fact was also created. The second schema change also produced another version of Sales. The query for the MTV approach should consider all these three versions of the fact to aggregate the yearly sales amount per city. Figure 4c shows the SQL query for the MTV approach. In case of HTV, only the new fact versions are created therefore, the query shown in Fig. 4b is simple as compared to the one for the MTV approach.

To make the user queries independent of the version management approach, we propose that each DW version is defined as set of views [12]. Figures 5a and 5b show the views defining the schema of level Store in version V_1 and V_2 in the STV and the HTV approaches, respectively. Since in case of the MTV approach a new table is created, the view definition will be different. Figure 5c shows the view defining the schema of level Store in version V_2. Obviously, the view definitions depend upon the approach used for storing the various versions (i.e., STV, MTV, or HTV), but once defined, such views may serve as virtual tables and thus user queries can be rewritten in term of such views [7,15] without considering the underlying the version management approach.

In addition to the versions defined as views, the system also maintains the metadata to support the coross-version queries. The metadata consists of the mappings between views, columns in the views, columns data types, and column names from one version to the other. In Fig. 6 we shows the partial metadata of our example MVDW. The mapping V_2 : Sales_V1 $\rightarrow$ Sales_V2 denotes that in version V_2, view Sales_V2 represents the same level that was represented by view Sales_V1. Similarly, the mapping V_3 : City_V1.Representative $\rightarrow$ City_V2.Manager denotes that in version V_3 column Representative of view City_V1 is renamed to Manager. Notice that this renaming resulted in the creation of a new version of level City and consequently a view City_V2. In our adopted numbering convention,

```
SELECT    AVG(Amount) as "Avg. Sales Amount", C.Name , T.Year
FROM      Sales S, Store E, City C, Time T
WHERE     S.TimeKey = T.TimeKey and S. StoreKey = E. StoreKey and E. City = C. CityKey or
          S. CityKey=C. CityKey
GROUP BY  C.Name, T.Year
```

```
WITH      SalesByCity AS (
SELECT    S.Amount, C.Name , T.Year FROM Sales_V1 S, Store E, City C, Time T
WHERE     S.TimeKey = T.TimeKey and S. StoreKey = E. StoreKey and E. City = C. CityKey
UNION ALL
SELECT    S2.Amount, C.Name , T.Year FROM Sales_V2 S2, City C, Time T
WHERE     S2.TimeKey = T.TimeKey and S2. CityKey = C.CityKey )
SELECT    AVG(Amount) as "Avg. Sales Amount", C.Name , T.Year FROM SalesByCity
GROUP BY  C.Name, T.Year
```

```
WITH      SalesByCity AS (
SELECT    S.Amount, C.Name , T.Year FROM Sales_V1 S, Store_V1 E, City C, Time T
WHERE     S.TimeKey = T.TimeKey and S. StoreKey = E. StoreKey and E. City = C. CityKey
UNION ALL
SELECT    S2.Amount, C.Name , T.Year FROM Sales_V2 S2, Store_V2 E2, Time T
WHERE     S2.TimeKey = T.TimeKey and S2.StoreKey = E2.StoreKey and E2.CityKey = C.CityKey
UNION ALL
SELECT    S3.Amount, C.Name , T.Year FROM Sales_V3 S3, City C, Time T
WHERE     S3.TimeKey = T.TimeKey and S3. CityKey = C.CityKey )
SELECT    AVG(Amount) as "Avg. Sales Amount", C.Name , T.Year FROM SalesByCity
GROUP BY  C.Name, T.Year
```

Fig. 4. Average sales amount per city per year: SQL queries for the STV, MTV, HTV approaches, respectively

the version number in the view is independent of the schema version number. Consider the query Q7 from Table 1, which computes total quantity of each product until today. Suppose that the user writes this query corresponding to current schema version but actually it requires data which was stored using all three schema versions of the DW. The schema mappings in the metadata provides the information that views Sales_V1, Sales_V2, and Sales_V3 represent the same fact Sales in all the schema versions, thus Q7 can be rewritten in term of these views.

Now consider query Q3 which requires total sales amount per customer and month from June 2014 till June 2015. The customer information exists only for the facts starting from January 2015. Ideally, the system should inform the user that though he/she asked the total sales for the customers from June 2014 , the answer is partial and it excludes the facts stored before January 2015 because dimension Customer did not exist in versions V_1 and V_2.

To support querying MVDWs we propose the architecture of a system shown in Fig. 7. As a first step to answer a cross-version query, the system determines the schema versions to be used. Either a user can explicitly specify a version(s) or they can be obtained from the time interval mentioned in the query. Depending upon the schema elements in the query and versions involved, the system determines whether the query is answerable, partially answerable, or not answerable at all. The metadata will be used to determine if there are any renamed

```
CREATE VIEW Store_V1 AS (
SELECT  StoreKey, Name, Area, CityKey
FROM    Store )
```

(a) Store in version V_1 using the STV, MTV, and HTV approaches

```
CREATE VIEW Store_V2 AS (
SELECT  StoreKey, Name, Manager, CityKey
FROM    Store )
```

(b) Store in version V_2 using the STV and HTV approaches

```
CREATE VIEW Store_V2 AS (
SELECT  StoreKey, Name, Manager, CityKey
FROM    Store_2 )
```

(c) Store in version V_2 using the MTV approach

Fig. 5. Views defining the schema of level Store in two schema versions

View Mappings	Column Mappings
V_2 : Sales_V1 → Sales_V2	V_2 : Sales_V1.Amount → Sales_V2.Amount
V_2 : Store_V1 → Store_V2	V_2 : . . .
V_2 : Time_V1 → Time_V1	V_3 : City_V1.Representative → City_V2.Manager
. . .	. . .

Fig. 6. Metadata of the example MVDW

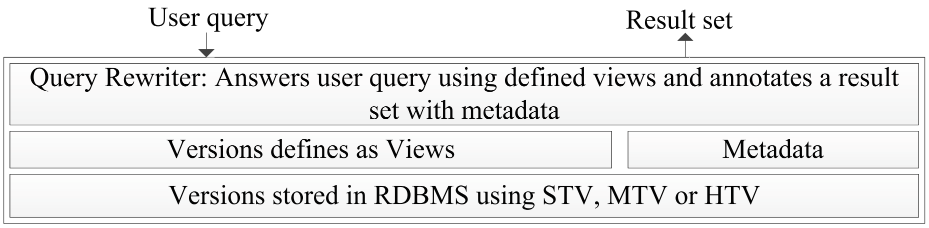

User query
Result set
Query Rewriter: Answers user query using defined views and annotates a result set with metadata
Versions defines as Views
Metadata
Versions stored in RDBMS using STV, MTV or HTV

Fig. 7. Proposed system to answer cross-version queries

attributes in the involved versions. Since each DW version is defined as a set of views, in the next step the user query is rewritten in term of the views belonging to the schema versions, which were determined in the first step. Finally, the query is executed and the result set is annotated with information such as missing or partial data.

6 Conclusions

The capability to maintain the history of changes in content and structure of a DW enables the user to recreate the state of the business world in the past or simulate the effect of a prospective change. Multiversion data warehouses (MVDWs) provide such capability by creating multiple schema and data versions. However, querying data from multiple schema versions is not a straightforward task. Further, the creation of each version may change the queries written for the already existing versions. In this paper, we discussed how changes in a multidimensional model impact on queries. Further, we showed that queries in a MVDW are dependent upon the version management approach. Since it would be convenient from the end user viewpoint to query a DW independently of the version management approach, we proposed a system to query data from the MVDWs.

One disadvantage of MVDWs is that they create multiple data versions to manage the content changes. This can further complicate the tasks of maintaining and querying a MVDWs, and may negatively impact performance of the system. One natural solution to the challenge of managing content and schema changes in a DW is to combine both, the temporal and multiversion data warehouses. As future work, we plan to combine the two approaches as a single solution which will be able to query multiple data versions stored in multiple schema versions. Further we want to present our solution as a data warehouse with temporal and multiversion functionality.

References

1. Ahmed, W., Zimányi, E., Wrembel, R.: A logical model for multiversion data warehouses. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2014. LNCS, vol. 8646, pp. 23–34. Springer, Heidelberg (2014)
2. Ahmed, W., Zimányi, E., Wrembel, R.: Temporal data warehouses: Logical models and querying. In: Proc. of EDA, pp. 33–47 (2015)
3. Curino, C., Moon, H.J., Deutsch, A., Zaniolo, C.: Automating the database schema evolution process. VLDB Journal **22**(1), 73–98 (2013)
4. Eder, J., Koncilia, C., Morzy, T.: The COMET metamodel for temporal data warehouses. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, p. 83. Springer, Heidelberg (2002)
5. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data & Knowledge Engineering **59**(2), 435–459 (2006)
6. Golfarelli, M., Rizzi, S.: A survey on temporal data warehousing. International Journal of Data Warehousing and Mining **5**(1), 1–17 (2009)
7. Halevy, A.Y.: Answering queries using views: A survey. VLDB Journal **10**(4), 270–294 (2001)
8. Huo, W., Tsotras, V.J.: Querying transaction–time databases under branched schema evolution. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part I. LNCS, vol. 7446, pp. 265–280. Springer, Heidelberg (2012)
9. Kaas, C., Pedersen, T.B., Rasmussen, B.: Schema evolution for stars and snowflakes. In: Proc. of ICEIS, pp. 425–433 (2004)
10. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. John Wiley & Sons (2013)
11. Malinowski, E., Zimányi, E.: A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. Data & Knowledge Engineering **64**(1), 101–133 (2008)
12. Medeiros, C.B., Bellosta, M., Jomier, G.: Multiversion views: Constructing views in a multiversion database. Data & Knowledge Engineering **33**(3), 277–306 (2000)
13. Moon, H.J., Curino, C., Ham, M., Zaniolo, C.: PRIMA: archiving and querying historical data with evolving schemas. In: Proc. of SIGMOD, pp. 1019–1022 (2009)

14. Roddick, J.F.: A survey of schema versioning issues for database systems. Information & Software Technology **37**(7), 383–393 (1995)
15. Srivastava, D., Dar, S., Jagadish, H.V., Levy, A.Y.: Answering queries with aggregation using views. In: Proc. of VLDB, pp. 318–329 (1996)
16. Wei, H.-C., Elmasri, R.: Schema versioning and database conversion techniques for bi-temporal databases. Annals of Mathematics and Artificial Intelligence **30**(1–4), 23–52 (2000)
17. Wrembel, R.: A survey on managing the evolution of data warehouses. International Journal of Data Warehousing & Mining **5**(2), 24–56 (2009)
18. Wrembel, R.: On handling the evolution of external data sources in a data warehouse architecture. In: Taniar, D., Chen, L. (eds.) Data Mining and Database Technologies: Innovative Approaches. IGI Group (2011)
19. Wrembel, R., Bębel, B.: Metadata management in a multiversion data warehouse. In: Meersman, R. (ed.) OTM 2005. LNCS, vol. 3761, pp. 1347–1364. Springer, Heidelberg (2005)