Temporal Data Warehouses: Logical Models and Querying

Waqas Ahmed*, Esteban Zimányi*, Robert Wrembel**

*Department of Computer and Decision Engineering (CoDE),
Université Libre de Bruxelles, Belgium
{waqas.ahmed, ezimanyi}@ulb.ac.be
**Institute of Computing Science,
Poznan University of Technology, Poland
robert.wrembel@cs.put.poznan.pl

Abstract. Data warehouses (DWs) integrate data from multiple and heterogeneous data sources. Most of the DW design methods assume that the contents of the dimensions in a DW will not change, but this is not the case in reality. Therefore, DWs must reflect these changes in the real-world in order to enable users to ask various types of temporal queries. Since temporal queries are complex and costly, it is necessary to know which modeling approach is better for such queries. In this paper, we discuss two possible approaches to implement a DW capable of maintaining the history of the changes in dimension members. We also present a classification of temporal queries that can be used to evaluate the two approaches.

1 Introduction

A data darehouse (DW) integrates data from multiple, heterogeneous, and often distributed external data sources (EDSs). Integrated data are then analyzed by means of multiple types of analytical applications. Data stored in a DW are modeled as multidimensional cubes that consist of facts and dimensions. A *fact* is a data item being analyzed. It includes numerical features called *measures* that quantify the fact. Some examples of facts include sales, cash withdrawals, and telephone calls. Facts are analyzed in the context of *dimensions* that typically have a hierarchical structure. Some examples of dimensions include Customer, Time, and Geography, the last one having a hierarchical structure City→State→Country. The instances of a dimension are called *members*. An example of a member of the Geography dimension is Brussels→Brussels-Capital Region→Belgium.

An inherent feature of EDSs is that they change their content and structures over time. Content changes result from daily business operations. Structural changes are a consequence of the need to support new business requirements, using new technologies, and changes in the legislation, to name a few. Both content and structural changes must be reflected in a DW. The maintenance of the history of changes allows users to inquire about the state of the business world at a given time. Moreover, for audit and accountability purposes, keeping an accurate historical context is often a business requirement.

Most DW design methods assume that the schema of a DW and the attribute values of dimension instances remain unchanged during the life cycle of a DW. However, this assumption turns out to be wrong in practice. Three alternative approaches have been proposed to handle the evolution of a DW. *Slowly changing dimensions* (SCDs) (Kimball and Ross, 2013) and *temporal data warehouses* (TDWs) (Golfarelli and Rizzi, 2009) (based on the research done in the field of temporal databases (Snodgrass and Ahn, 1986)) handle the changes in the content of DWs, whereas *multiversion data wsarehouses* (MVDWs) (Ahmed et al., 2014; Wrembel and Bębel, 2007) handle both the content and structural changes. SCDs focus on maintaining the history of changes in dimension members. TDWs provide built-in support for storing and querying time-varying facts and dimension members. MVDWs maintain the history of changes in the content and structure by means of DW versions.

SCDs, being an intuitive and easy-to-implement approach, is becoming recognized by industry (e.g., SQL Server support for SCDs [1]). On the other hand, temporal support has been incorporated into the SQL standard (Kulkarni and Michels, 2012) and is implemented in some DBMSs, e.g., IBM DB2 [2], Teradata (Al-Kateb et al., 2013), and PostgreSQL [3].

When a DW keeps historical information, a user may ask various types of queries about the evolution of the business world. Answering such queries is not trivial as identifying and manipulating the records valid during the period of interest can be very complex and resource demanding. Therefore, while designing a DW that maintains the history of its evolving states, it would be helpful for a designer to know which of the aforementioned modeling approaches is the most suitable for a given application scenario. For this reason, in this paper we focus on comparing type 2 SCDs with temporal data warehouses presented in Malinowski and Zimányi (2008), with respect to complexity of queries.

In particular, the **contributions** of this paper are the following:

- we discuss two approaches to implement a DW capable of tracking history,
- we present a classification of temporal queries and describe the operators for implementing them, and
- we briefly compare the query performance in the two approaches.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Sect. 3, we introduce a running example and its conceptual schema using the MultiDim model. In Sect. 4, we present two possible approaches for implementing the running example. In Sect. 5 we present a classification of business queries and discuss the temporal operators needed to implement such queries. In Sect. 6 we discuss with the help of an example query the performance of the two possible implementation approaches. Finally, Sect. 7 concludes the paper and provides considerations for future research.

2 Related Work

In this section, we present the work related to temporal database benchmarking, query classification for DWs, and performance evaluation of modeling techniques for evolving DWs.

1. http://msdn.microsoft.com/en-us/library/ms141715.aspx
2. http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/
3. http://pgxn.org/dist/temporal_tables/

Dunham et al. (1995) presented a benchmark model for bitemporal databases. The authors identified various components of the benchmark as user requirements, temporal dataset, and queries to be run. They also provided an extensive list of queries with different variants to test various performance aspects. OLTP-Bench (Difallah et al., 2013) is a benchmark for bitemporal relational databases which allows users to choose a workload from the synthetic DB benchmarks and real-world web applications. Han and Lu (2014) highlighted the potential research issues in big data benchmarking.

Kaufmann et al. (2013) classified the temporal queries into three classes: time travel, temporal joins, and temporal aggregates. Furthermore, they presented an implementation of various temporal operators for an in-memory DBMS. Although many contributions have been made for benchmarking temporal databases, a comprehensive benchmark for DWs with temporal functionality is still awaited.

TSQL2 (Snodgrass, 1995) was proposed as a query language for bitemporal databases but temporal features did not appear in the SQL standard until the release of SQL-2011 (Kulkarni and Michels, 2012). SQL-2011 provides support for a number of temporal features such as period datatype, temporal primary keys, temporal referential integrity constraints, temporal predicates, valid time and transaction time, temporal inserts, updates, and deletes. However, the support for temporal operators like temporal coalescing, temporal joins, and temporal aggregation (Snodgrass, 2000; Zimányi, 2006) has not been included in the standard. Faisal and Sarwar (2014) presented a classification of queries based on the input and output attributes of the query and studied the performance of type 2 and type 6 SCDs.

Kaufmann et al. (2014) extended the TPC-H for bitemporal databases and tested the performance of the temporal features of various commercial database management systems. They also provided a query taxonomy for the benchmark.

3 Preliminaries

In this section, we introduce a motivating example which we will use throughout the rest of the paper. It consists of a sales DW in which the user needs to maintain the historical context for some dimensions so that she could query the historical states of the business world. To represent the DW conceptually, we use the temporal extension of the MultiDim model. A complete description of the model and mapping rules to transform the model into various logical schemas can be found in (Malinowski and Zimányi, 2008; Vaisman and Zimányi, 2014). To make this paper self contained, we give below a brief description of the temporal MultiDim model.

The model allows the user to model various temporal constructs such as temporal attributes, temporal levels, and temporal relationships. A MultiDim schema consists of *fact* relationships which contain numeric attributes called *measures* and a discrete group of alphanumeric attributes called *levels* which form *dimensions*. Instances of a level are called *members*. In the temporal MultiDim model, the history of changes is recorded by associating orthogonal time dimensions with the changing constructs. A *temporal attribute* keeps track of the changes in its value and the time when these changes occur. A *temporal level* is a level for which the application requires to store a time frame associated with its members. A *temporal relationship* between two levels keeps record of the time frame for which the members of the child levels are linked to the members of their parent levels.

The temporal MultiDim model supports various temporality types including valid time, transaction time, and lifespan. *Valid time* represents the time when a data record is considered valid in a business world. *Transaction time* represents the time when a data record is contained in the system. *Lifespan* temporality is associated with levels and is used to keep track of the existence of the members as a whole. The lifespan of an object *o* can be seen as the valid time of the related fact, '*o exists*'. A level with a lifespan can have any number of attributes with non-temporal, valid time, and/or transaction time temporality. For the sake of simplicity, in this paper, we only consider a schema with a lifespan and valid time temporality.

Motivating Example Figure 1 shows a sales DW which is modeled using the MultiDim model. The schema consists of a central fact Sales which is linked to dimensions Customer, Employee, Product, Geography, and Time. Sales is used to analyze measures UnitPrice, Discount, SalesAmount, Freight, and NetAmount. The symbols '+!'and '/'represent non-additive and derived measures, respectively. Notice that the Time dimension is linked to Sales, twice and provides two different perspectives to analyze measures, namely, OrderDate and DueDate. Such dimensions are called *role-playing* dimensions. In our example DW, the user

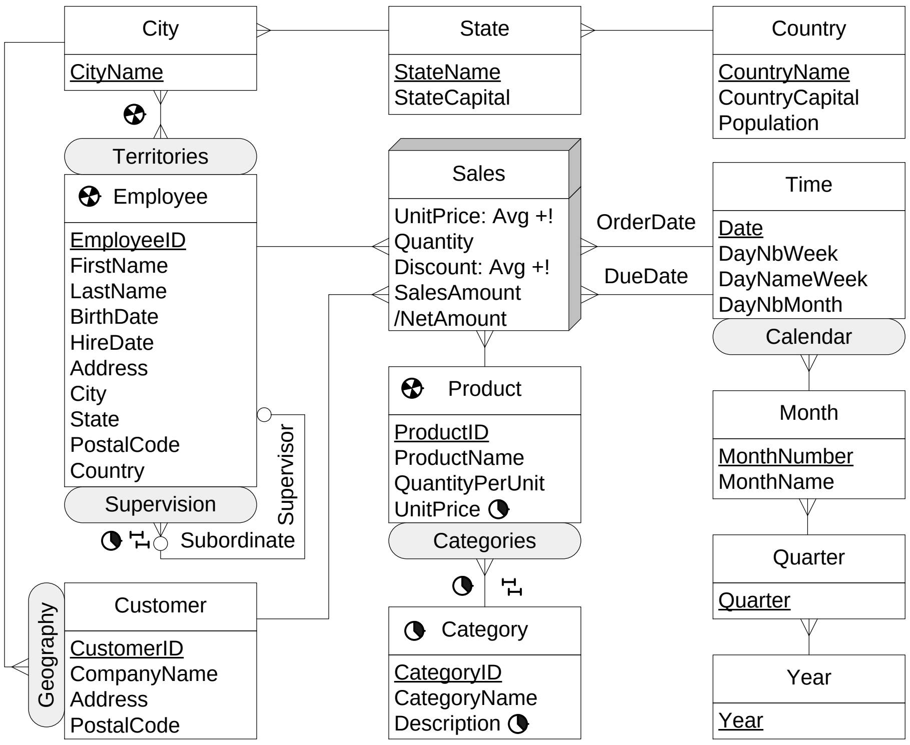


FIG. 1: MultiDim model of the sales data warehouse

has the following temporal requirements :

R1: A product may be discontinued and become unavailable for selling. A discontinued product may be reintroduced for selling at a later time. The period during which a product was available for sales should be maintained;

R2: Product prices change very often due to promotions or discounts. A history of changes in a product's unit price should be maintained;

R3: From time to time, a product may be reassigned to a different category. The history of changes in a product category should be maintained;

R4: The description of a category may also change. The history of the changes in a category's description should be maintained.

R5: Employees may leave the company and rejoin later. The time for which an employee has been working for the company should be maintained.

R6: Employees may be reassigned to work under different supervisors. The history of changes of an employee's supervisor should be maintained.

R7: An employee is assigned to work in more than one city and more than one employee work in a single city. The time for which an employee has been working in a city should be maintained.

The above requirements are represented in Fig. 1 as follows. The lifespan of employees, products, and categories are represented by associating a lifespan temporality with their levels, as shown by the symbols ☢ and ◑, which represent, respectively, a discontinuous and a continuous lifespan. The symbol ◑ next to the attributes UnitPrice and Description belonging to levels Product and Category means that the history of changes in the value of these attributes should be maintained. The symbols in the Product-Category, Employee-Supervisor, and Employee-City relationships imply that the history of the assignment of a child members to parent members should be maintained. The symbol ⊓ in these relationships means that the validity period of the assignment should be contained in the validity of the child and parent members. For example, when assigning a category to a product, the validity of the assignment should be included in the validity period of the product and the category.

4 Logical Implementations

In this section, we explain the two possible approaches to implement the conceptual schema shown in Fig. 1, namely type 2 SCDs and temporal data warehouses (TDWs).

4.1 Type 2 Slowly Changing Dimensions

Kimball and Ross (Kimball and Ross, 2013) argued that dimension attributes are not static and they slowly change in time. They referred to such dimensions as slowly changing dimensions and provided seven techniques to track the changes in attribute values. Below, we briefly describe the three basic types of SCDs.

- Type 1: Changes are handled by overwriting the existing data, thus, no history is maintained.
- Type 2: A new level member version is created for each change in an attribute value. A flag value or an independent time dimension (two attributes representing the start time

and end time of the validity of the version) is used to represent the current level member version. Using type 2 SCDs, an unlimited number of versions of a level member can be created, but to uniquely identify each version, a surrogate key is required.

– Type 3: A separate attribute is added to capture the history for each changing attribute. Whenever an attribute's value changes, its existing value is recorded in the separate attribute and the new value overwrites the existing value. Type 3 SCD provides limited support for history, but is useful in what-if analysis scenarios.

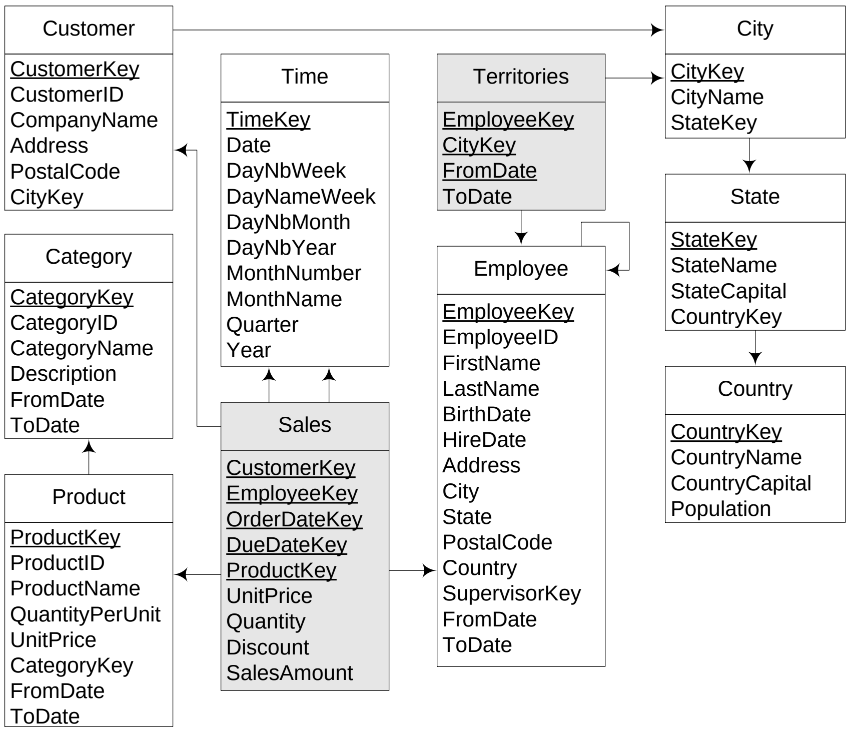
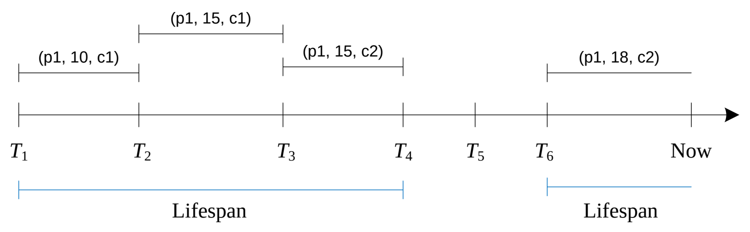


FIG. 2: Type 2 SCD implementation of the sales data warehouse in Fig. 1

For our example schema, we can achieve the user requirements pertaining to maintaining the history by implementing our conceptual schema with type 2 SCDs, as shown in Figure 2. This implementation fulfills the user requirements mentioned above as follows. To meet requirements R1, R2, and R3, two attributes FromDate and ToDate are added to level Product. These attributes represent a closed-open interval during which a product record is considered valid. The value of attribute ToDate of the currently valid record is set to a special value 'Now'. Although level product has ProductID as business key, but to identify the various versions of the same product, a surrogate key is required that is why, attribute ProductKey is also added to the level and serves as its primary key. The same technique is applied to meet requirements R4, R5, and R6. Since there is a many to many relationship between level Employee and City, a new table Territories is created to record the cities assigned to an employee and employees working in a city. To fulfill R7, an interval is also included in Territory table which depicts the duration for which an employee has been working in a particular city. As there is no need to create the versions of records belonging to the table, a surrogate key is not introduced. Attributes EmployeeKey, CityKey, and FromDate combined together can serve as the primary key of the table Territory.



(a) Time line of the evolution of unit price and category of product p1

Product

ProductKey	ProductID	UnitPrice	CategoryKey	FromDate	ToDate
1	p1	10	c1	T_1	T_2
2	p1	15	c1	T_2	T_3
3	p1	15	c2	T_3	T_4
4	p1	18	c2	T_5	Now

(b) Type 2 SCD implementation

ProductLifespan

ProductKey	FromDate	ToDate
p1	T_1	T_4
p1	T_6	Now

ProductUnitPrice

ProductKey	UnitPrice	FromDate	ToDate
p1	10	T_1	T_2
p1	15	T_2	T_4
p1	18	T_6	Now

ProductCategory

ProductKey	CategoryKey	FromDate	ToDate
p1	c1	T_1	T_3
p1	c2	T_3	T_4
p1	c2	T_6	Now

(c) Temporal data warehouse (TDW) implementation

FIG. 3: Keeping the evolution of product p1 in the data warehouse

To further understand type 2 SCDs, let us consider the scenario depicted in Fig. 3a, which involves maintaining the lifespan of a level member, the history of changes in an attribute value, and a temporal relationship. Suppose product p1, with a unit price of 10, was introduced at time T_1 and was assigned to category c1. At T_2, its unit price was changed from 10 to 15 and at T_3 it was assigned to category c2. At T_4 the product was discontinued and was no longer offered for selling. At T_6, the same product p1 was reintroduced and it is available for selling until today. The type 2 SCD implementation of this scenario is shown in Fig. 3b. For brevity, in these figures we only show the changing attributes.

4.2 Temporal Data Warehouses

Malinowski and Zimányi (2008) introduced a different approach to implement the MultiDim model into the relational one. They proposed the following mapping rules to map temporal levels, temporal attributes, and temporal relationships to ER constructs. We assume that each temporality type represents a set of intervals, which is a typical case in a DW.

1. Each temporal level with a lifespan is represented by the member and a multivalued composite attribute to represent the lifespan of the member of the level.
2. Each temporal attribute of a level is represented by the value of attribute and a multivalued composite attribute to represent the validity period of the value.
3. Each temporal relationship between two levels is represented by the related child-parent members and a multivalued composite attribute representing the period for which the relationship was valid.

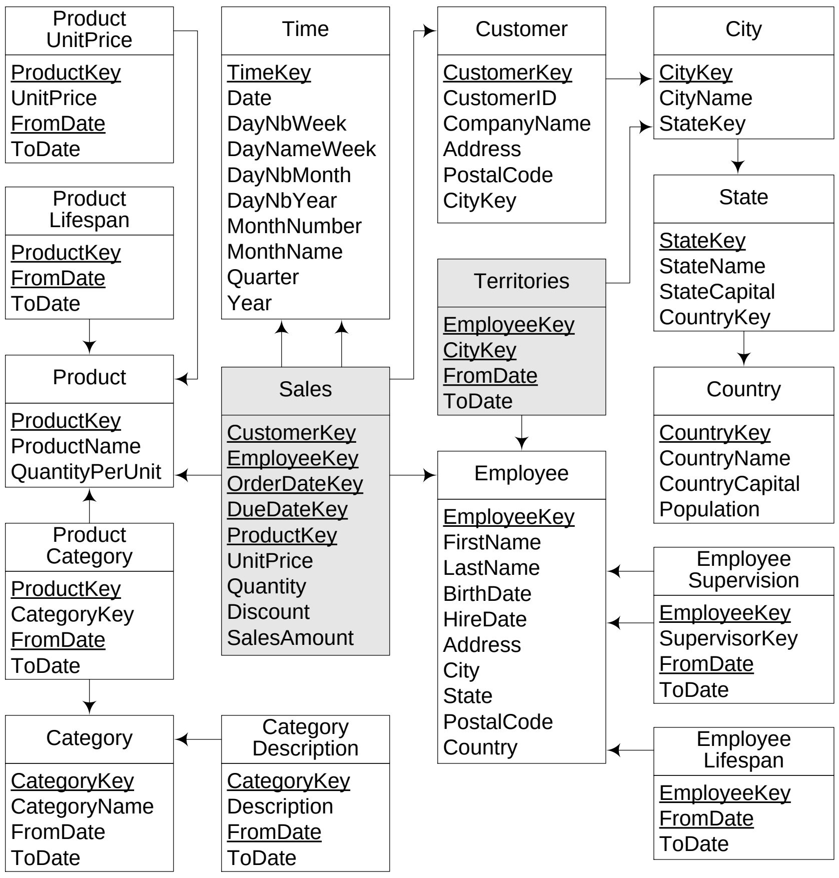


FIG. 4: Temporal data warehouse implementation of the sales data warehouse in Fig. 1

Figure 4 shows the TDW implementation of the schema given in Fig. 1. This implementation fulfills the user requirements as follows. For R1 and R5, respectively, tables ProductLifespan and EmployeeLifespan are created which hold the identifier of the member whose lifespan is to be maintained and a pair of attributes FromDate and ToDate. For requirements R2, a table ProductUnitPrice is created with product identifier, its unit price, and two attributes FromDate and ToDate. The same mapping rule is applied to track the changes in a category description, i.e., R4. To keep the history of product-category assignments, i.e., R3, a table

No.	Class	Example
Q1	Temporal roll-up	Total quantity per category and month.
Q2	Temporal-roll-up (with window)	Monthly year-to-date sales per category.
Q3	Temporal roll-up (recursive)	Total sales amount made by an employee and her subordinates during 1997.
Q4	Temporal selection	For each employee, total sales amounts of products she sold with unit price greater than $30 at the time of the sale.
Q5	Temporal projection	Total sales amount for supervisors.
Q6	Temporal union	Total sales amounts for products assigned to categories beverages or dairy products.
Q7	Temporal join	Name, unit price, and total sales amount by month for products.
Q8	Temporal difference	Total sales amount for employees assigned to only one city.
Q9	Temporal aggregation	For each month, give the total quantity of products sold per category.
Q10	Temporal aggregation (for a many-to-many relationship)	For each employee, total sales amount, number of cities, and number of states to which she is assigned.
Q11	Temporal aggregation (duration of interval)	For each pair of employee and supervisor, total number of days when the supervision lasted.
Q12	Temporal universal quantifier	Total sales for categories in which all products have a price greater than $7
Q13	Time slice	Average unit price by category as of January 1st, 1997.

TAB. 1: Classes of temporal business queries and example queries

ProductCategory is created with the product identifier, the category identifier, and the period during which the assignment was valid. The mapping for employee-city assignments is the same as for type 2 SCDs.

Figure 3c shows the TDW implementation of the scenario described above for tracking the evolution of product p1. Notice that instead of storing all product versions in a single table, every evolving attribute is stored in a separate table. Moreover, no surrogate keys are introduced and the combination of the business key and FromDate serve as the primary key of the table.

5 Temporal Business Questions

In this section, we describe the temporal operators needed to query historical data. For this, we use the classification of temporal business queries shown in Tab. 1.

The **temporal roll-up** operator involves summarizing data from a lower level to an upper level in a hierarchy. It differs from the traditional roll-up in the sense that it must take into account the time interval during which a child-parent relationship is valid. Q1 is an example, which requires to summarize quantities per category and month. Since the assignment of products to categories varies in time, we must ensure that every product is rolled-up to the category that it was assigned to at the *order date*. This is illustrated in Fig. 5a, which shows, from top to bottom, the aggregation of the Time dimension to the Month level, for products P1, P2, and P3, their assignment to categories C1 and C2 as well as Quantity measures from their Sales, and the result of the temporal roll-up. Q2 and Q3 are two variants of the temporal roll-up which require year-to-date and recursive roll-up operators, respectively.

The **temporal selection** operator keeps tuples that satisfy some condition. It is similar to the traditional selection, excepted that the selection predicate may include a condition on the time interval during which a temporal attribute or a temporal link is valid. Q4 is an example, where the selection predicate requires that only those products that had a unit price greater than $30 at the time of sales are selected.

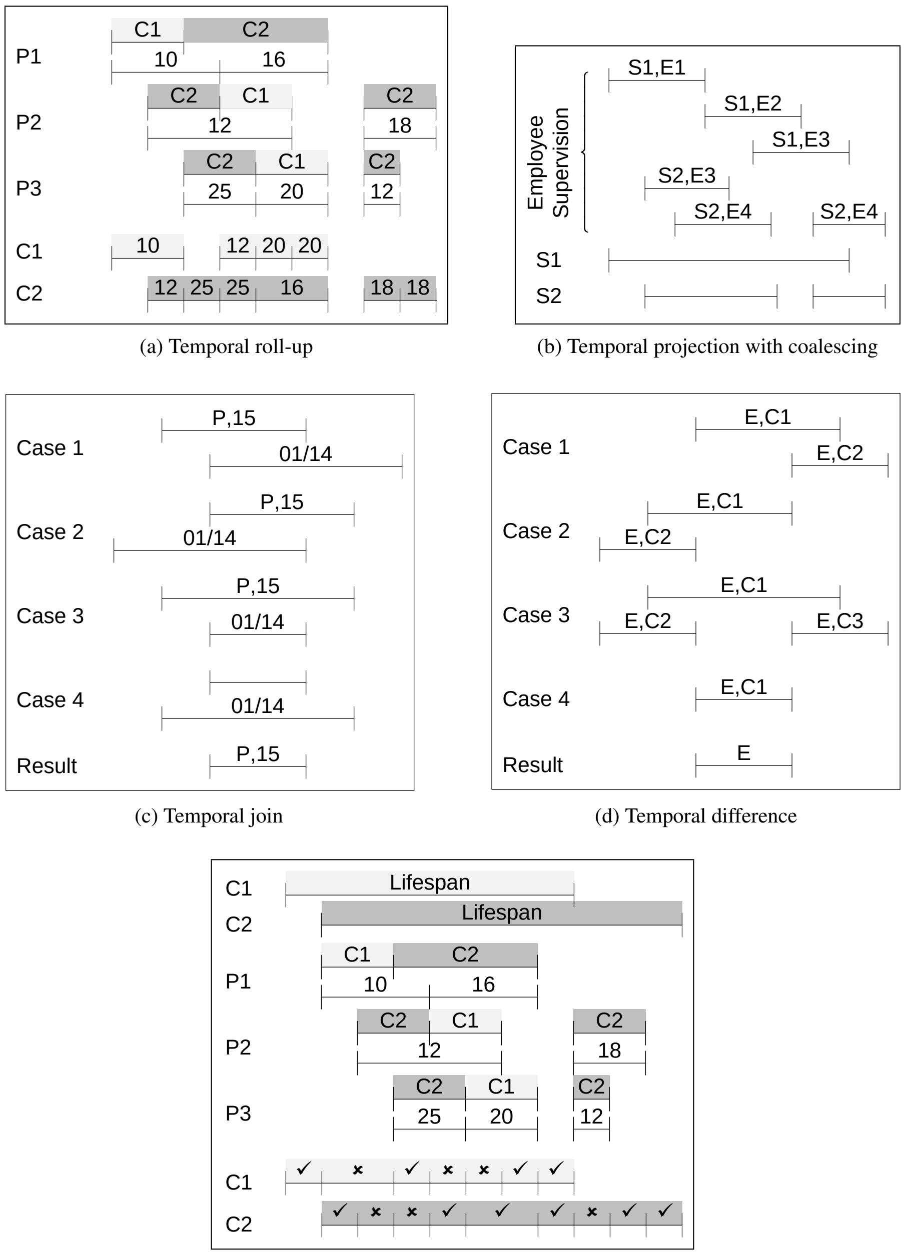
The **temporal projection** opearation returns a set of computed attributes with an associated time interval from a relation. Often this time interval is derived from the time interval associated to each tuple in the temporal table. Q5 is an example of a temporal projection query. Since the employee-supervisor assignment varies in time, we first need to determine the periods during which an employee was supervisor of at least one employee. Then, we must coalesce these periods to obtain the total period of an employee as supervisor, and finally, compute the total sales for this period.

The **temporal union** operator is equivalent to the traditional relational set operator and returns the set union of two relations. Q6 is an example, which asks for total sales amounts for products assigned to categories beverages or dairy products. Since the assignment of the products to categories varies in time, first we need to compute the intervals during which a product was assigned to one of the categories requested. Then, these intervals must be coalesced to obtain the total interval for each product-category assignment. Finally, we can compute the total sales for the products during the periods they were assigned to one of the categories requested.

The **temporal join** operator merges the tuples of two relations if a join condition is met and the joined records overlap during a time period. Q7 is an example, which asks for the name, unit price, and total sales amount by month for products. Even the query asks total sales by month, since a product's unit price changes in time, first we must split months according to the variation of unit price. While joining the sales with the product, we must join it with the unit price valid at that the *order date*.

The **temporal difference** operator is typically used to filter out information from a superset of potential answers. Q8 is an example, which requires to compute the total sales amount for the employees assigned to only one city. Once obtained the employees using temporal difference, we can coalesce the each employee record and then may compute the sales whose order date belongs to the period in which an employee is assigned to a single city. Then, we can group the results by employee to compute the total sales amount.

The **temporal aggregation** operator involves partitioning the timeline, grouping the tuples into these partitions, and then computing aggregates for these groups. Q9 is an example, which for each month, computes the total quantity of products sold per category. To do so, firstly, we need to compute the period associated with each month and then compute the periods during



(a) Temporal roll-up

(b) Temporal projection with coalescing

(c) Temporal join

(d) Temporal difference

(e) Temporal universal quantification

FIG. 5: Temporal operators

which the number of products by category is constant. Secondly, we have to split the periods by month. Thirdly, we need to compute the total quantity of the products by category for each of the periods. Finally, the main query coalesces the computed periods to obtain the final results. Q10 and Q11 are also temporal aggregation queries, which compute aggregates for temporal many-to-many relationships and according to a duration of time, respectively.

The **temporal universal quantifier** operator is used for verifying that a condition is satisfied *for all* tuples in a set. Q12 is an example, which asks total sales for categories in which all products have a price greater than $7. To compute the result set, we need first to obtain, for each category, the prices of its products with their associated interval and the days on which there is a change in unit price or category assignment of one of its products. After converting these days into periods, we can select the tuples where there is no product in the category with unit price less than or equal to $7 in the interval. Finally, after coalescing these tuples we can compute, for the each coalesced tuple, the total sales amount for the category that occurred during the interval.

The **timeslice** operator returns the tuples of relation valid at a given time.Timeslice reconstructs the state of a relation at a given point in time whereas, in temporal selection, the condition may involve any time granularity i.e., a point in time, an interval, or a period etc. Q13 is a query which requires to compute the average unit price by category as of January 1, 1997. To compute the average unit price, we need to extract the product and category tuples which were valid at the given time.

6 Querying Temporal Data Warehouses

In this section, we compare the type 2 SCD and the TDW implementations given in Figs. 2 and 4, respectively, with respect to querying. Consider query Q7 above: *Name, unit price, and total sales amount by month for products*. We suppose that the following view, which computes the start and end date of months using a closed-open representation, has been created

```
CREATE VIEW Month(Year, MonthNumber, FromDate, ToDate) AS (
          SELECT     Year, MonthNumber, MIN(Date), DateAdd(month, 1, MIN(Date))
          FROM       Time
          GROUP BY Year, MonthNumber )
```

The type 2 SCD version of the above query is as follows:

```
WITH ProdUnitPrice(ProductID, UnitPrice, FromDate, ToDate) AS (
             SELECT     ProductID, UnitPrice, FromDate, ToDate
             FROM       Product ),
      ProdUnitPriceCoalesced AS (
             -- Coalescing the table ProdUnitPrice above ... ),
SELECT       P.ProductName, U.UnitPrice, SUM(SalesAmount) AS SalesAmount,
             dbo.MaxDate(M.FromDate, U.FromDate) AS FromDate,
             dbo.MinDate(M.ToDate, U.ToDate) AS ToDate
FROM         Sales S, Time T, Product P, ProdUnitPriceCoalesced U, Month M
WHERE        S.OrderDateKey = T.TimeKey AND S.ProductKey = P.ProductKey AND
             P.ProductID = U.ProductID AND dbo.MaxDate(M.FromDate, U.FromDate) <
             dbo.MinDate(M.ToDate, U.ToDate) AND
             dbo.MaxDate(M.FromDate, U.FromDate) <= T.Date AND
             T.Date < dbo.MinDate(M.ToDate, U.ToDate)
```

```
GROUP BY P.ProductName, U.UnitPrice, M.FromDate, U.FromDate,
         M.ToDate, U.ToDate
ORDER BY P.ProductName, dbo.MaxDate(M.FromDate, U.FromDate)
```

Even if the query asks total sales by month, the months must be split according to the variation of unit price for products. This version of the query requires a temporal projection (with coalescing) of the Product table in order to compute the ProductUnitPrice table that is available in the TDW version. This is done in the two temporary tables ProdUnitPrice and ProdUnitPriceCoalesced. Then, the main query performs a temporal join of the latter table with the Month view, and a traditional join with the remaining tables, prior to do the aggregation of the SalesAmount measure. For this, the functions MinDate and MaxDate are used, which compute, respectively, the minimum and maximum date of the two arguments.

On the other hand, the TDW version of this query is as follows:

```
SELECT   P.ProductName, U.UnitPrice, SUM(SalesAmount) AS SalesAmount,
         dbo.MaxDate(M.FromDate, U.FromDate) AS FromDate,
         dbo.MinDate(M.ToDate, U.ToDate) AS ToDate
FROM     Sales S, Time T, Product P, ProductUnitPrice U, Month M
WHERE    S.OrderDateKey = T.TimeKey AND S.ProductKey = P.ProductKey AND
         P.ProductKey = U.ProductKey AND dbo.MaxDate(M.FromDate, U.FromDate) <
         dbo.MinDate(M.ToDate, U.ToDate) AND
         dbo.MaxDate(M.FromDate, U.FromDate) <= T.Date AND
         T.Date < dbo.MinDate(M.ToDate, U.ToDate)
GROUP BY P.ProductName, U.UnitPrice, M.FromDate, U.FromDate, M.ToDate, U.ToDate
ORDER BY P.ProductName, dbo.MaxDate(M.FromDate, U.FromDate)
```

This query performs a temporal join of the Month view and the ProductUnitPrice table and a traditional join of the other tables prior to computing the total sales amount in the periods obtained. As can be seen, the query for the type 2 SCD implementation is much more complex than the one for the TDW version. As we have said, temporal projection with coalescing is a costly operator.

Cconsider now query Q13 above: *Average unit price by category as of January 1st, 1997*. For the type 2 SCD implementation, since the evolution of products is kept in a single table, the corresponding query would need a join of two tables (i.e., Product and Category), whereas in the corresponding query for the TDW implementation a join of four tables (i.e., Product, ProductUnitPrice, ProductCategory, and Category) is needed.

7 Conclusion and Future Work

Data warehouse (DW) users may want to preserve historical data so that they could query the various status of the business world. A user may ask several types of business queries pertaining to historical data. In this paper, we described two approaches, type 2 slowly changing dimensions and temporal data warehouses, to model a DW capable of keeping history. We further provided a classification of the temporal business queries. As it is estimated that in a data warehouse 80% of the resources are consumed by 20% of the queries, it is very important to know which of these two modeling approaches is most suitable for time-related business queries. In the future, we have plans to test both approaches at different data scales and determine which one is suitable for each class of queries.

Acknowledgements This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence Doctoral College" (IT4BI-DC). Additionally, the work of Robert Wrembel has been supported by the Polish National Science Center (NCN), grant No. 2011/01/B/ST6/05169.

References

Ahmed, W., E. Zimányi, and R. Wrembel (2014). A logical model for multiversion data warehouses. In *Proc. of DaWaK 2014*, Number 8646 in LNCS, pp. 23–34. Springer.

Al-Kateb, M., A. Ghazal, A. Crolotte, R. Bhashyam, J. Chimanchode, and S. P. Pakala (2013). Temporal query processing in Teradata. In *Proc. of EDBT 2013*, pp. 573–578. ACM.

Difallah, D. E., A. Pavlo, C. Curino, and P. Cudre-Mauroux (2013). OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment 7*(4), 277–288.

Dunham, M. H., R. Elmasri, M. A. Nascimento, and M. Sobol (1995). Benchmarking temporal databases: A research agenda. Technical report, Department of Computer Science and Engineering, Southern Methodist University.

Faisal, S. and M. Sarwar (2014). Handling slowly changing dimensions in data warehouses. *Journal of Systems and Software 94*, 151–160.

Golfarelli, M. and S. Rizzi (2009). A survey on temporal data warehousing. *International Journal of Data Warehousing and Mining 5*(1), 1–17.

Han, R. and X. Lu (2014). On big data benchmarking. arXiv preprint arXiv:1402.5194.

Kaufmann, M., P. M. Fischer, N. May, and D. Kossmann (2014). Benchmarking bitemporal database systems: Ready for the future or stuck in the past? In *Proc. of EDBT 2014*, pp. 738–749.

Kaufmann, M., P. Vagenas, P. M. Fischer, D. Kossmann, and F. Färber (2013). Comprehensive and interactive temporal query processing with SAP HANA. *Proceedings of the VLDB Endowment 6*(12), 1210–1213.

Kimball, R. and M. Ross (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. John Wiley & Sons.

Kulkarni, K. and J.-E. Michels (2012). Temporal features in SQL:2011. *ACM SIGMOD Record 41*(3), 34–43.

Malinowski, E. and E. Zimányi (2008). 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.

Snodgrass, R. T. (Ed.) (1995). *The TSQL2 Temporal Query Language*. Kluwer.

Snodgrass, R. T. (2000). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann.

Snodgrass, R. T. and I. Ahn (1986). Temporal databases. *IEEE Computer 19*(9), 35–42.

Vaisman, A. and E. Zimányi (2014). *Data Warehouse Systems: Design and Implementation*. Springer.

Wrembel, R. and B. Bębel (2007). Metadata management in a multiversion data warehouse. In *Journal of Data Semantics VIII*, Number 3761 in LNCS, pp. 118–157. Springer.

Zimányi, E. (2006). Temporal aggregates and temporal universal quantification in standard SQL. *ACM SIGMOD Record 35*(2), 16–21.

Résumé

Les entrepôts de données (EDs) intègrent les données provenant de multiples sources de données hétérogènes. La plupart de méthodes de conception d'entrepôts de données supposent que le contenu des dimensions dans un entrepôt n'est pas modifié, mais ceci n'est pas le cas en réalité. Par conséquent, les EDs doivent refléter ces changements dans le monde réel pour permettre aux utilisateurs de demander différents types de requêtes temporelles. Souvent, ces requêtes sont complexes et coûteuses, et par conséquent, il est nécessaire de savoir quelle approche de modélisation est meilleure pour ces requêtes. Dans cet article, nous discutons deux approches possibles pour mettre en oeuvre un ED capable de maintenir l'histoire de l'évolution des membres des dimensions. Nous présentons également une classification des requêtes temporelles qui peut être utilisé pour évaluer ces deux approches.