JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 1213-1241 (2022)
DOI: 10.6688/JISE.202211_38(6).0007

Spatiotemporal Data Warehousing for Event Tracking Applications*

FRANK S. C. TSENG[1,+] AND ANNIE Y. H. CHOU[2]
[1]*Department of Information Management*
National Kaohsiung University of Science and Technology
Kaohsiung, 824 Taiwan
E-mail: imfrank@nkust.edu.tw
[2]*Department of Computer and Information Science*
ROC Military Academy
Kaohsiung, 830 Taiwan
E-mail: imyhchou@gmail.com

In this paper, we propose a multidimensional spatiotemporal modeling framework of data warehouse creation for tracing dynamic events in contemporary applications, like crowd contact tracing for Covid-19 prevention. Such a framework offers a natural and consistent solution for slowly changing dimension management. It provides a progressive evolution from traditional static data management to modern dynamic data analysis with spatiotemporal tracking capabilities for IoT applications. Based on such a framework, entity-centered resource integration and related business intelligence applications can be rigorously developed, managed and properly tracked.

Keywords: big data, data warehouse, digital footprint, dimensional modeling, slowly changing dimension, spatiotemporal tracking, temporal database

1. INTRODUCTION

Data warehouse (DW) [5, 13] is a concept regarding a central repository of integrated data for big data analytics. Defined by Inmon [13], there are four characteristics of a DW; *i.e.*, subject-oriented, integrated, time-variant, and non-volatile collection of data in support of the decision-making process for business operations. Generally, a DW is created from one or more disparate sources, together with historical transformations and summary aggregations. It has been considered as the kernel technology of *business intelligence* (BI) and big data analytics, used for creating multidimensional on-line analytical processing (OLAP) reports for managers throughout enterprises to assist contemporary administrative decision-makings for more than two decades.

Today's cloud applications, like social networking or Covid-19 prevention apps, gain continuous interests in the public domain. The inspired power thrives enough to lead the trend of our community. Such phenomenon inspires contemporary applications to gradually move their focus from monitoring traditional static relationships to tracking dynamic relationships, which tend to be rapidly changing, especially when the target of analysis is the moving trajectory or status changing of objects or persons, as studied in [3, 11]. Should

Received October 25, 2021; revised December 21, 2021; accepted February 9, 2022.
Communicated by De-Nian Yang.
* This research was partially supported by the Ministry of Science and Technology, Taiwan, under contract No. MOST 109-2410-H-992-023-MY2.
+ Corresponding author.

the derived data be properly collected, analyzed, and utilized, it would be impossible to overestimate the benefits and influence for deriving intelligence from the masses [23]. As users today are drowning in data, but thirsty for pursuing interested or needed knowledge from every kind of digital footprints generated by humans or entities.

Therefore, we are aware that it needs a spatiotemporal dimensional modeling framework for recording entity- or user-centered behaviors and tracking the data interweaved by networks, crowds and markets; and reasoning over a highly connected world [5]. Based on the well-developed relational technologies, it is becoming inevitable to consider adding spatial and temporal features into the elements of data management and data warehousing methodologies. We believe such a framework can help us develop a structure with the following merits:

1. In the era of Internet-of-Things (IoT), one can track the time and space state changes of specific items, objects or vehicles for various applications; integrate the related information; or predict their future trends.
2. Netizens can integrate their own digital footprints from participating online social media for personal information integration based on their own timelines.
3. The spatial and temporal states of disseminated information can be integrated to make the circulation of information more transparent, and be critically traced back to the original initiator based on the information provenance (such that infections of Covid-19, fake news or disinformation dissemination can be mitigated).

In Fig. 1, we depict a simple star schema (dimensional modeling of a data warehouse [13]) with two temporal dimension tables to illustrate such concept, where temporal dimension tables, D_1 and D_2, are modeled as a series of changing relations D_i^{tj}, each has their content status respectively corresponding to t_j (*e.g.*, t_1, t_2 and t_3). The fact table F is used to store the integrated digital footprints merged from different data sources or social networks.

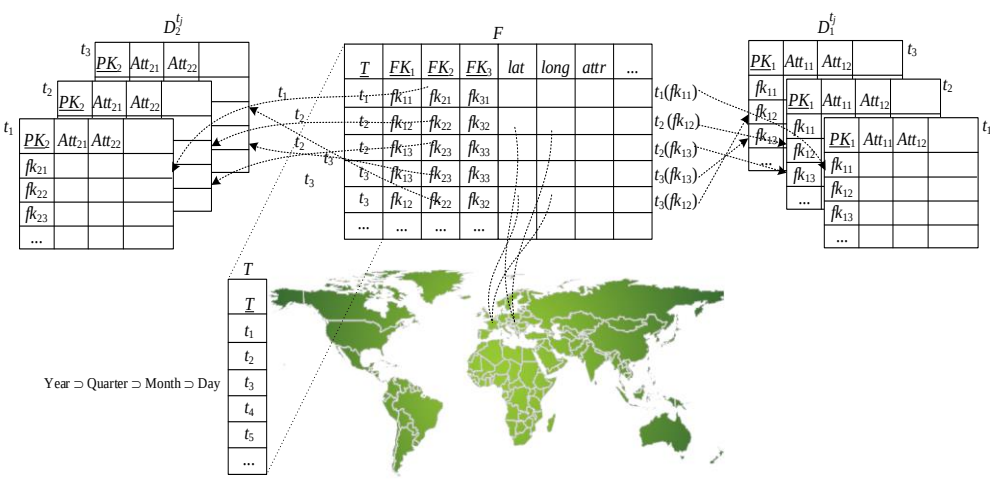

Fig. 1. Spatiotemporal multidimensional data modeling.

2. RELATED WORKS

To construct a data warehouse (DW), a dimensional modeling process may be conducted to create cubes from several dimensions based on the data correlated in relational tables [5, 13-15]. In a DW, the indispensable needs are tracking the timing of changes in dimension attributes, together with related business activities, to create business intelligence reports more accurately. That is why a *time* dimension is usually involved in the process of constructing a multidimensional cube. Over the past two decades, the research fields regarding data warehouse and temporal database [6, 18, 19] have emerged into the concept of temporal data warehouse (TDW) [9].

Besides, by including spatial data into consideration, Vaisman and Zimanyi [26] presented a conceptual framework using an extensible data type system to define a spatiotemporal data warehouse. They mainly focused on the data representation and the expressive power of related queries; proposed a taxonomy of different classes of queries of increasing expressive power; and showed how to express these queries using an extension of the tuple-oriented relational calculus with aggregate functions.

Based on the survey of Golfarelli and Rizzi [10], many tools or theories have been developed to support time-varying data management, such as slowly changing dimension (SCD) management [13], manipulation of temporal queries [19], temporal view materialization and TDW modeling [9]. They distinguish these researches into the following categories:

1. *Handling changes in the data warehouse*: regarding the work of maintaining the data synchronization between the data sources and the data warehouse.
2. *Handling data changes in the data mart*: concerning the data inconsistent problems when data are continuously imported into data marts, as the instances of dimensions and hierarchies may not always be static.
3. *Handling schema changes in the data mart*: handling the situations of schema structure changes in response to the evolving business demands, such that new levels or measures may be added, while others may become obsolete.
4. *Temporal data queries*: paying specific attention to querying temporal data, when schema may change.
5. *Temporal data warehouse design*: conceptually designing a temporal data warehouse based on some ad hoc approaches to handle different kinds of temporal data [6].

Although these approaches effectively remedy parts of the problems in the ETL (Extraction, Transformation, and Loading) process [2, 20], TDWs or spatiotemporal data warehouses, users may still miss important trends in data or infer decisions ineffectively, as there is no general framework to consistently capture the geo-varying and time-varying nature of data, such that data can be properly managed in transaction systems, and be correctly analyzed in data warehouse systems [1, 2].

We noticed that the most common excruciations in maintaining cubes is the *slowly changing dimensions* (SCDs) management in temporal considerations. Even though there are many types of methodologies have been proposed to deal with issues involving SCDs

management [14], the processing of SCDs poses significant challenges to cube maintenance and the ways to derive precise business intelligence; and users still suffer from their complexities and cumbersomeness [19].

Kimball & Ross (2002) [15] identified eight types of SCD and proposed some maintenance techniques for each type as follows:

1. *Type 0: Retain Original*: Some dimension attribute values are regarded as unchangeable, like the color of a human's eye or skin tone, and facts are always grouped by the original value. There is no need for SCD management.
2. *Type 1: Overwrite*: the old attribute value in the dimension row is overwritten with the new value to reflect the most recent assignment, and therefore destroy the update history. Such type can be conducted for those attributes which won't affect the ordinary business processes if they change. For example, if someone's birth date changes: an update to this immutable value is most likely a data entry error (or a correction to an earlier error). Therefore, the entire change history of such an attribute value does not need to be kept.
3. *Type 2: Add a New Row*: add a new row in the dimension table with the updated attribute values for recalling the update history.
4. *Type 3: Add a New Attribute*: add a new attribute in the dimension table to preserve the old attribute value, used for recalling the update history.
5. *Type 4: Add a Mini-Dimension*: when a group of attributes in a dimension rapidly changes and is split off into a *mini-dimension*. This situation is sometimes called a *rapidly changing monster dimension*. This method is advantageous for highly volatile or frequently used attributes in a very large size dimension. Hence, if we know there are attributes that fit this category, then just split it off into its own physical mini-dimension table. The surrogate keys of both tables are captured as foreign keys to the fact table. An example is separating age band or income level attributes from a customer base dimension and including in the mini-dimension table.
6. *Type 5: Add a Mini-Dimension and Type 1 Outrigger*: used to accurately preserve historical attribute values, plus report historical facts according to current attribute values.
7. *Type 6: Add Type 1 Attributes to Type 2 Dimension*: delivers both historical and current dimension attribute values.
8. *Type 7: Dual Type 1 and Type 2 Dimensions*: a hybrid technique used to support both as-was and as-is reporting.
9. *Type N: Preserving a Complete History of Changes in a Nested Relation*: a newly technique developed by Garani *et al.* [10], which needs no additional relations, columns or rows, and requires no extra join operations or surrogate keys.

We realize that the most important mission of managing SCDs is the reflection of historical events. If the time-varying nature of data cannot be properly managed in transaction systems, users may miss important trends in data or may infer the trends incorrectly, and the real-time situations may not be correctly analyzed in data warehouse systems [1]. Santos & Belo (2011) [19] believe that correctly dealing with the timing changes is a critical success factor to the deployment of a data warehouse system. However, they pointed out that classifying SCDs into types is actually inappropriate, as these types are applied to attributes, and we could have dimension tables with some attributes belonging to type 1,

others belonging to Types 2 or 3. Therefore, they have analyzed the proposed SCD types and concluded that separating current data from historical data is the only feasible approach in terms of performance and simplicity of design and implementation. But they did not provide a feasible approach on this matter.

Therefore, we intend to present a general framework in this study, by extending our previous work [25], to naturally resolve the complexity of the management of SCDs. The approach mainly focuses on the basic elements of multidimensional modeling for a spatiotemporal data warehouse, such that the entity-centered resource integration and spatiotemporal business intelligence applications can be rigorously developed, managed and properly tracked.

Our paper is organized as follows. We propose a spatiotemporal multidimensional modeling framework of data warehouse for tracing events in various applications in Section 3. The basic elements of our framework are formally defined in Section 4. Some practical applications will be discussed in Section 5. Finally, we summarize and conclude our work in Section 6.

3. A SPATIOTEMPORAL DATA WAREHOUSE FRAMEWORK

Spatiotemporal analysis helps us describe a phenomenon in a certain location and period of time – for example, shipping movements across a geographic area over time. By envisioning how objects move in space and time, one can use spatial-temporal reasoning methods to solve multi-step problems [18]. The analysis can produce different results depending on how space is defined (by zip code, a census tract or a state). Time can also provide answers in different granularities, depending on whether it is measured in second, minute, hour, day, month or year.

Snodgrass [21] distinguished time into the following three types:

1. *User-defined time*: an uninterpreted time value (to be defined and handled by the system developers).
2. *Valid time*: it captures the history of a changing reality (when a fact was true in the modeled real world).
3. *Transaction time*: it records the sequence of states of a changing relation (when a fact was recorded in the database).

To simplify our discussion, we ignore the user-defined time and suppose a tuple is valid starting from the transaction time to the next update or deletion. That is, the concepts of valid time and transaction time are unified into one type in the following. As these kinds of time are orthogonal, a table can be associated with none, one, two, or even all these kinds of time [21]. Therefore, our proposed model can be extended accordingly by adding different types of time as other dimensions.

3.1 Preliminaries of Temporal Relation

Traditional database systems primarily process the "current" data, *i.e.*, no matter what CRUD operations have been posed, the systems always override the old data and keep the new data values only. Such databases are also called *transient databases* [10], they only

store the current representation of real-world objects without any track of changes being logged, and there is no way to recall what the object was in the past. Under such circumstances, the entire data changing history will be lost.

However, in practical applications, keeping the status and relationship changings for stored entities, based on different timing, is inseparable from our daily life. For example, to prevent the outbreak of COVID-19 effectively, it needs to gather crowd information in databases, with timing and geolocations, through people's cell phones. Besides, grasping each customer's status changing is also commonly employed in one-to-one marketing activities.

Conceptually, a traditional relational database is regarded as a collection of flat tables with two-dimensional data structure (although a relation is actually of one-dimension with *n* degrees). To properly manage the timing of data status changes, a relation in a database needs to be treated as a three-dimensional table, with *time* as the third dimension. A database with such challenging multi-version data processing capabilities is called a *temporal database* or *historical database* [18]. Practically, the key points of temporal data management for data warehousing based on our approach are as follows:

1. Building a user-defined function *D*_state(*@time*) and its related procedures (or triggers) for any traditional relation *D* (which is supposed to be used for dimension creation), such that the third dimension *time* can be sliced using the parameter *@time*, to retrieve the correct status of *R* at time *@time*.
2. Joining the correct versions of all dimensions, namely D_i_state(*@time*), with the fact table, based on the transaction time (represented as a parameter *@time*) of each fact record to retrieve the correct status of all dimensions.
3. Using the joined result for multidimensional on-line analytical processing (OLAP).

Snodgrass [21] has disclosed the fundamental elements of developing time-oriented database applications simply using SQL and presents many SQL techniques for a variety of complex applications. Following the advocating of Snodgrass [21], the primary features of SQL: 2011 standard (or ISO/IEC 9075:2011) [16, 29] are focus on the improvements of supporting new capabilities for temporal databases.

To track entity status in a database, we need to keep a complete record about the time of creating an entity, the time of updating attribute values, and the time of deleting an entity. Suppose we have a Bookstores table as follows:

```
Create Table Bookstores (
        no int not null,
        name nchar(30) not null,
        rank tinyint not null,
        region nchar(30) not null,
        city nchar(30) not null,
        primary key (no))
```

To trace the access history of Bookstores, Snodgrass [21] suggests creating another table, namely Bookstores_Log, to log every access record with the following structure:

```
Create Table Bookstores_Log (
    auto_key int identity not null,
    no int not null,
```

```
name nchar(30) null,
rank tinyint not null,
region nchar(30) null,
city nchar(30) null,
when_changed datetime2(7) not null,
who_changed char(10) not null,
operation char(1) not null check (operation in ('I', 'D', 'S', 'U')),
primary key (auto_key, no, when_changed))
```

The attributes in boldface are newly added in addition to all the attributes of Bookstores. Note that the primary key in Bookstores_Log now becomes a composite key (***auto_key***, *no*, *when_changed*), and all the attributes in Bookstores should be included, other than the original primary key attribute, must be set to allow null. Then, when records are inserted/deleted or attribute values are updated in Bookstores, three triggers (respectively for insert, delete and update) should be prepared to proactively log the related details in Bookstores_Log. We modify the approach developed by Snodgrass [21], and construct these triggers in Transact-SQL (of Microsoft SQL Server) as listed in Fig. 2:

```
Create Trigger Bookstores_Insert on Bookstores for INSERT
As
Insert into Bookstores_Log (no, name, rank, region, city, when_changed, who_changed, operation)
                    select no, name, rank, region, city, getdate(), suser_sname(), 'I' from inserted
--
Create Trigger Bookstores_Delete on Bookstores for DELETE
As
Insert into Bookstores_Log (no, name, rank, region, city, when_changed, who_changed, operation)
                    select no, name, rank, region, city, getdate(), suser_sname(), 'D' from deleted
--
Create Trigger Bookstores_Update on Bookstores for UPDATE
As
Insert into Bookstores_Log (no, name, rank, region, city, when_changed, who_changed, operation)
              select no, name, rank, region, city, getdate(), suser_sname(), 'D' from deleted
Insert into Bookstores_Log (no, name, rank, region, city, when_changed, who_changed, operation)
              select no, name, rank, region, city, getdate(), suser_sname(), 'I' from inserted
```

Fig. 2. Triggers to be created for proactively log the related details in Bookstores_Log.

Next, we need a reconstruction function to return the status of the Bookstores table at a certain time. We use SQL Server's user-defined functions to implement it by adopting the SQL structure recommended by Snodgrass [21]. In Fig. 3, we illustrate the function Bookstores_State(@requested_date) returning the state of Bookstores at the time specified by @requested_date.

```
Create function Bookstores_State(@requested_date datetime) Returns TABLE
as RETURN( Select no, name, rank, region, city
            From Bookstores_Log AS E1
            Where E1.when_changed =
                        (Select max(E2.when_changed)
                         From Bookstores_Log AS E2
                         Where E1.no = E2.no and E2.when_changed < @requested_date)
                         And E1.operation <> 'D' and E1.operation <> 'S'))
```

Fig. 3. Bookstores_State(@requested_date) returns the state of the time @requested_date.

Now, the function Bookstores_State(@requested_date) can be regarded as a three dimensional version of Bookstores, with the third dimension *time*, and it returns the content of Bookstores at any time specified by @requested_date. We illustrate this concept in Fig. 4, where the content of table Bookstores at time t_1 (returned by Bookstores_State(t_1)) is shown at the bottom. At time t_2, the following updates were performed:

1. the *name*s of the second and fifth tuples were respectively changed from 'Tsutaya' to 'Kinokuniya' and 'MCA Store' to 'Pubu'; and
2. the *city* of the first tuple was changed from 'Taipei' to 'Kaohsiung'.

At time t_3, two tuples (the sixth and seventh tuples) were inserted, and the top relation will be returned by Bookstores_State(t_3). If there are no further update operations, the top relation is the content of Bookstores after time t_3. That is, users can obtain any past status of the Bookstores by calling the function Bookstores_State(@requested_date).

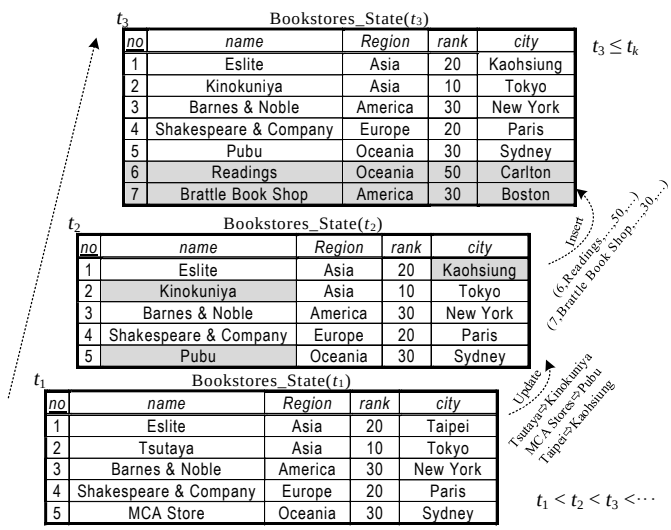

Fig. 4. The Function Bookstores_State(@requested_date) is a 3-dimensional version of Bookstores.

Besides, in addition to this function, we can also create a view to retrieve all the records that have appeared in Bookstores, as well as their life cycles. The SQL statement is elaborated in Fig. 5.

```
Create View Bookstores_lifecycle (no, name, rank, region, city, start_date, stop_date)
As
Select E1.no, E1.name, E1.rank, E1.region, E1.city, E1.when_changed, E2.when_changed
From Bookstores_Log AS E1, Bookstores_Log AS E2
Where E1.no = E2.no and E1.when_changed < E2.when_changed and E1.operation <> 'D'
and NOT EXISTS (Select * From Bookstores_Log AS E3
                Where E1.no = E3.no and E1.when_changed < E3.when_changed
                and E3.when_changed < E2.when_changed)
Union
Select no, name, rank, region, city, when_changed, getdate() From Bookstores_Log AS E1
Where E1.operation <> 'D' and NOT EXISTS (Select * From Bookstores_Log AS E3
                                          Where E1.no = E3.no
                                            and E1.when_changed < E3.when_changed)
```

Fig. 5. Defining the view Bookstores_ lifecycle for Bookstores.

By following the scenarios in Fig. 4, we experimentally inserting the following tuples at '2021-09-28 14:32:27' into Bookstores:

id	*name*	*region*	*rank*	*City*
1	Eslite	Asia	20	Taipei
2	Tsutaya	Asia	10	Tokyo
3	Barnes & Noble	America	30	New York
4	Shakespeare & Company	Europe	20	Paris
5	MCA Store	Oceania	30	Sydney

Then update attribute values of the first, second, and fifth tuples as described in Fig. 4 at '2021-09-28 14:40:58':

id	*name*	*region*	*rank*	*City*
1	Eslite	Asia	20	Kaohsiung
2	Kinokuniya	Asia	10	Tokyo
3	Barnes & Noble	America	30	New York
4	Shakespeare & Company	Europe	20	Paris
5	Pubu	Oceania	30	Sydney

Finally, insert two tuples (the sixth and seventh tuples) at '2021-09-28 14:43:18'.

id	*name*	*region*	*rank*	*City*
6	Readings	Oceania	50	Carlton
7	Brattle Book Shop	America	30	Boston

After completion, the result returned by the SQL statement "select * from Bookstores_lifecycle" at '2021-09-28 15:10:39' is as Fig. 6 lists. The content of Bookstores_Log is as Fig. 7 depicts.

id	*name*	*region*	*rank*	*City*	*Start_date*	*Stop_date*
1	Eslite	Asia	20	Kaohsiung	2021-09-28 14:40:58	2021-09-28 15:10:39
1	Eslite	Asia	20	Taipei	2021-09-28 14:32:27	2021-09-28 14:40:58
2	Kinokuniya	Asia	10	Tokyo	2021-09-28 14:40:58	2021-09-28 15:10:39
2	Tsutaya	Asia	10	Tokyo	2021-09-28 14:32:27	2021-09-28 14:40:58
3	Barnes & Noble	America	30	New York	2021-09-28 14:32:27	2021-09-28 15:10:39
4	Shakespeare & Company	Europe	20	Paris	2021-09-28 14:32:27	2021-09-28 15:10:39
5	MCA Store	Oceania	30	Sydney	2021-09-28 14:32:27	2021-09-28 14:40:58
5	Pubu	Oceania	30	Sydney	2021-09-28 14:40:58	2021-09-28 15:10:39
6	Readings	Oceania	50	Carlton	2021-09-28 14:43:18	2021-09-28 15:10:39
7	Brattle Book Shop	America	30	Boston	2021-09-28 14:43:18	2021-09-28 15:10:39

Fig. 6. The result returned by the SQL "select * from Bookstores_lifecycle".

Auto_key	*no*	*name*	*region*	*rank*	*city*	*when_changed*	*who_changed*	*operation*
1	5	MCA Store	Oceania	30	Sydney	2021-09-28 14:32:27	sa	I
2	4	Shakespeare & Company	Europe	20	Paris	2021-09-28 14:32:27	sa	I
3	3	Barnes & Noble	America	30	New York	2021-09-28 14:32:27	sa	I
4	2	Tsutaya	Asia	10	Tokyo	2021-09-28 14:32:27	sa	I
5	1	Eslite	Asia	20	Taipei	2021-09-28 14:32:27	sa	I
8	1	Eslite	Asia	20	Taipei	2021-09-28 14:40:58	sa	D
9	1	Eslite	Asia	20	Kaohsiung	2021-09-28 14:40:58	sa	I
10	2	Tsutaya	Asia	10	Tokyo	2021-09-28 14:40:58	sa	D
11	2	Kinokuniya	Asia	10	Tokyo	2021-09-28 14:40:58	sa	I
12	5	MCA Store	Oceania	30	Sydney	2021-09-28 14:40:58	sa	D
13	5	Pubu	Oceania	30	Sydney	2021-09-28 14:40:58	sa	I
14	6	Readings	Oceania	50	Carlton	2021-09-28 14:43:18	sa	I
15	7	Brattle Book Shop	America	30	Boston	2021-09-28 14:43:18	sa	I

Fig. 7. The content of Bookstores_Log.

Note that in Fig. 6, a tuple with a *stop_date* of '2021-09-28 15:10:39' (*i.e.*, the time the SQL query was issued, can be obtained by selecting max(stop_date)) indicates that the tuple currently exists in Bookstores. Therefore, we can generate the current content of Bookstores relation from the following query posed on Bookstores_lifecycle:

```
select no, name, region, rank, city
from Bookstores_lifecycle
where stop_date >= (select max(stop_date) from Bookstores_lifecycle)
```

The returned result is exactly the same as the result of issuing the following statement:

```
select no, name, region, rank, city from Bookstores
```

3.2 A Demonstrative Example

To explain the importance of utilizing temporal tables to keep updated history for precise data analysis, we present a demonstrative example in the following. This example is based on three tables:

1. Bookstores(*no*, *name*, *region*, *rank*, *city*), with primary key (*no*)
2. Orders(*tid*, *no*, *id*, quantity), where *no* and *id* are foreign keys reference Bookstores.*no* and Books.*id*, respectively. Its primary key is a composite attribute (*tid*, *no*, *id*).
3. Books(*id*, *bookname*, *category*, *author*, *price*, *publisher*), with primary key (*id*).

Based on the status changes of Fig. 8, suppose there are four time spots, namely t_1, t_2, t_3 and t_4. Before t_1, suppose the content of relations Bookstores and Books are as listed above the dotted line indicated by t_1; *i.e.*, on top of the diagram, there are 5 bookstores and 6 books. Right at t_1, there are 17 orders inserted into Orders. Then, after t_1, the following SQL statements have been executed:

```
update Bookstores set city = 'Kaohsiung' where no = 1;
update Bookstores set name = 'Kinokuniya' where no = 2;
update Bookstores set name = 'Pubu' where no = 5;
update Books set publisher = 'Oxford University' where id = 1;
```

After executing these statements, the contents of Bookstores and Books are as listed below t_1, with new attribute values shown in gray areas. Then, things happen at t_2, t_3 and t_4, which we explain as follows:

1. The following SQL statements are executed at t_2. (Two bookstores are inserted):
   ```
   insert into Bookstores (no, name, region, rank, city)
   values (6, 'Readings', 'Oceania', 50, 'Carlton'), (7, 'Brattle Book Shop', 'America', 30, 'Boston')
   ```
2. The following SQL statements are executed at t_3 (Five orders are inserted into Orders):
   ```
   insert into Orders (tid, no, id, quantity) values (getutcdate(), 1, 1, 60), (getutcdate(), 2, 1, 70),
   (getutcdate(), 6, 3, 80), (getutcdate(), 7, 3, 30), (getutcdate(), 7, 5, 50)
   ```
3. The following SQL statements are executed at t_4 (Two orders are inserted into Orders):
   ```
   insert into Orders (tid, no, id, quantity) values (getutcdate(), 5, 6, 15), (getutcdate(), 5, 1, 25)
   ```

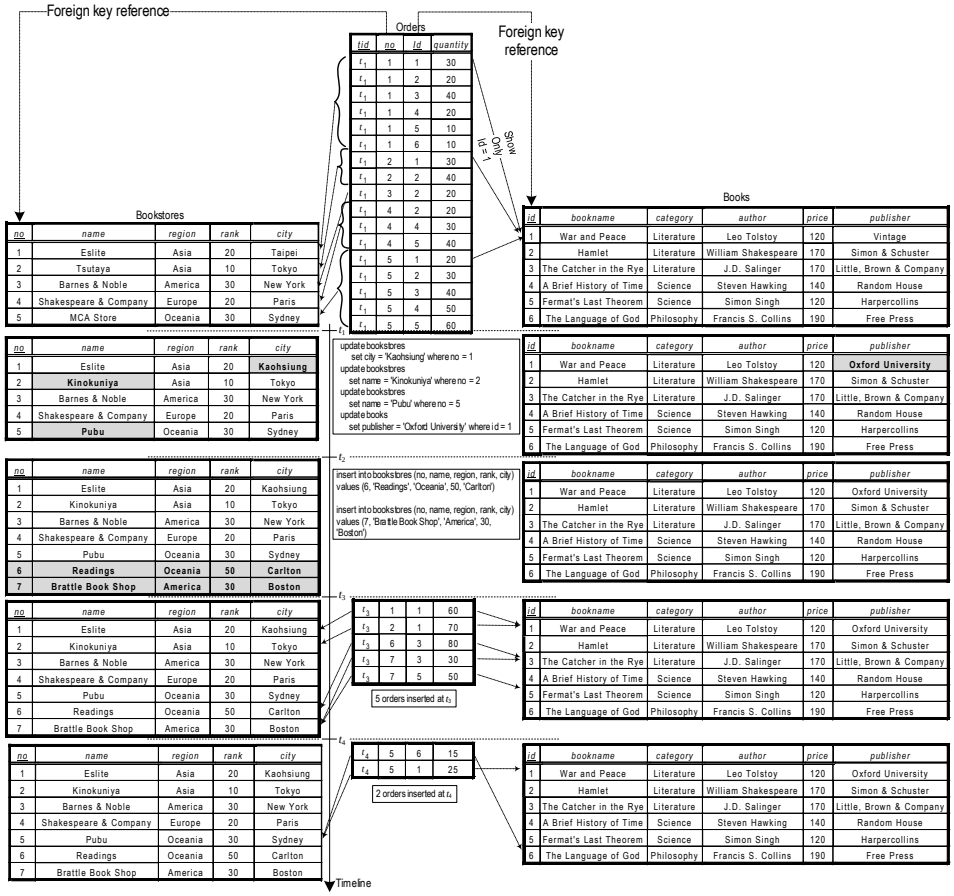

Fig. 8. A demonstrative example.

After executing these statements, the contents of Bookstores and Books are listed below t_2, t_3 and t_4, at the left-hand-side and right-hand-side, respectively, with new attribute values shown in gray areas. For relation Orders, we gradually extend its content to show its data evolution. Based on the timeline, the foreign key reference relationships are linked as shown by the corresponding arrows. That means, for example, the following two tuples links to the same tuples of Bookstores and Books, but with different attribute values at different time spots of t_1 and t_4 (attribute values are shown in boldface to show the differences):

1. On the top of Fig. 8, the fifth tuple from the last, *i.e.*, (t_1, 5, 1, 20), in relation Orders represents an order issued by the bookstore
   (5, '**MCA Store**', 'Oceania', 30, 'Sydney')
   with the book
   (1, 'War and Peace', 'Literature', 'Leo Tolstoy', 120, '**Vintage**'),
   based on the status of Bookstores and Books at t_1.
2. On the bottom of Fig. 8, the newly inserted tuple (t_4, 5, 1, 25) in Orders represents an order sent by the bookstore

(5, '**Pubu**', 'Oceania', 30, 'Sydney')

with the book

(1, 'War and Peace', 'Literature', 'Leo Tolstoy', 120, '**Oxford University**'),

based on the content of Bookstores and Books at t_4.

Traditionally, if we build a cube without using temporal tables for dimensions, then the contents of dimension tables Bookstores and Books, together with the fact table Orders and related foreign key references; *i.e.*, the database status after t_4, can be shown as Fig. 9 depicts. Some data changing histories are lost in such a situation.

no	name	region	rank	city
1	Eslite	Asia	20	Kaohsiung
2	Kinokuniya	Asia	10	Tokyo
3	Barnes & Noble	America	30	New York
4	Shakespeare & Company	Europe	20	Paris
5	Pubu	Oceania	30	Sydney
6	Readings	Oceania	50	Carlton
7	Brattle Book Shop	America	30	Boston

id	bookname	category	author	price	publisher
1	War and Peace	Literature	Leo Tolstoy	120	Oxford University
2	Hamlet	Literature	William Shakespeare	170	Simon & Schuster
3	The Catcher in the Rye	Literature	J.D. Salinger	170	Little, Brown & Company
4	A Brief History of Time	Science	Steven Hawking	140	Random House
5	Fermat's Last Theorem	Science	Simon Singh	120	Harpercollins
6	The Language of God	Philosophy	Francis S. Collins	190	Free Press

tid	no	Id	quantity
t_1	1	1	30
t_1	1	2	20
t_1	1	3	40
t_1	1	4	20
t_1	1	5	10
t_1	1	6	10
t_1	2	1	30
t_1	2	2	40
t_1	3	2	20
t_1	4	2	20
t_1	4	4	30
t_1	4	5	40
t_1	5	1	20
t_1	5	2	30
t_1	5	3	40
t_1	5	4	50
t_1	5	5	60
t_3	1	1	60
t_3	2	1	70
t_3	6	3	80
t_3	7	3	30
t_3	7	5	50
t_4	1	1	60
t_4	7	5	50

Fig. 9. The status of an example database with related reference key references after t_4.

Therefore, any on-line analytical processing (OLAP) would get imprecise result due to the loss of status changing histories of all dimension tables. For instance, by employing the pivot table technique of Excel to analyze the data in Fig. 9, using "*publisher* ⊃ *bookname*" as the row hierarchy, and "*region* ⊃ *city* ⊃ *bookname*" as the column hierarchy, we obtain the result shown in Fig. 10 (to make the result more concise, we hide some sub-aggregations). Such report reveals only part of the history and should be regarded as distorted and ineffective, as some facts are lost. We list just a few examples in the following:

1. The order (t_1, 1, 1, 30) issued at t_1 actually comes from the bookstore 'Eslite' located in 'Taipei'. But we cannot find any 'Taipei'-related information in this report.
2. The orders (t_1, 2, 1, 30) and (t_3, 2, 1, 70) respectively issued at t_1 and t_3 should be separated, as they come from different bookstore names, namely 'Tsutaya' and 'Kinokuniya'. But they are all aggregated into 'Kinokuniya', and we cannot find any 'Tsutaya'-related information in this report.
3. Similarly, the orders (t_1, 5, 1, 20) and (t_4, 5, 1, 25) should be separated based on the bookstores named 'MCA Stores' and 'Pubu', respectively. But they are all aggregated into 'Pubu', and all 'MCA Store'-related information is lost in this report.

	Free Press	Harpercollins	Little, Brown and Company	Oxford University	Random House	Simon & Schuster	Total
	The Language of God	Fermat's Last Theorem	The Catcher in the Rye	War and Peace	A Brief History of Time	Hamlet	
America		**50**	**30**			**20**	**100**
Boston		**50**	**30**				**80**
Brattle Book Shop		50	30				80
New York						**20**	**20**
Barnes & Noble						20	20
Asia	**10**	**10**	**40**	**190**	**20**	**60**	**330**
Kaohsiung	**10**	**10**	**40**	**90**	**20**	**20**	**190**
Eslite	10	10	40	90	20	20	190
Tokyo				**100**		**40**	**140**
Kinokuniya				100		40	140
Europe		**40**			**30**	**20**	**90**
Paris		**40**			**30**	**20**	**90**
Shakespeare & Company		40			30	20	90
Oceania	**15**	**60**	**120**	**45**	**50**	**30**	**320**
Carlton			**80**				**80**
Readings			80				80
Sydney	**15**	**60**	**40**	**45**	**50**	**30**	**240**
Pubu	15	60	40	45	50	30	240
Total	**25**	**160**	**190**	**235**	**100**	**130**	**840**

Fig. 10. A distorted and ineffective report for the demonstrative example.

	Free Press	Harpercollins	Little, Brown and Company	Oxford University	Random House	Simon & Schuster	Vintage	Total
	The Language of God	Fermat's Last Theorem	The Catcher in the Rye	War and Peace	A Brief History of Time	Hamlet	War and Peace	
America		**50**	**30**			**20**		**100**
Boston		**50**	**30**					**80**
Brattle Book Shop		50	30					80
New York						**20**		**20**
Barnes & Noble						20		20
Asia	**10**	**10**	**40**	130	**20**	**60**	60	**330**
Kaohsiung				60				60
Eslite				60				60
Taipei	**10**	**10**	**40**		**20**	**20**	30	**130**
Eslite	10	10	40		20	20	30	130
Tokyo				70		40	30	**140**
Kinokuniya				70				70
Tsutaya						40	30	70
Europe		**40**			**30**	**20**		**90**
Paris		**40**			**30**	**20**		**90**
Shakespeare & Company		40			30	20		90
Oceania	**15**	**60**	**120**	25	**50**	**30**	20	**320**
Carlton			**80**					**80**
Readings			80					80
Sydney	**15**	**60**	**40**	25	**50**	**30**	20	**240**
MCA Store		60	40		50	30	20	200
Pubu	15			25				40
Total	**25**	**160**	**190**	155	**100**	**130**	80	**840**

Fig. 11. A precise report of the demonstrative example.

In Fig. 11, we present a meaningful report using the same aggregated temporal hierarchies, and use gray areas to highlight the primary different parts between this report and Fig. 10.

Such a report precisely retains all the data changing histories, and we can switch the analytical result back to any time spot instantly, by introducing the Time dimension (*i.e.*, the *tid*, which includes all time spots) as a pivot table slicer. In Fig. 12, we list the sliced snapshots (a), (b) and (c) respectively for t_1, t_3, and t_4. These reports honestly reflect the correct situations for our analysis.

	Free Press	Harpercollins	Little, Brown and Company	Random House	Simon & Schuster	Vintage	Total
	The Language of God	Fermat's Last Theorem	The Catcher in the Rye	A Brief History of Time	Hamlet	War and Peace	
America					**20**		**20**
New York					**20**		**20**
Barnes & Noble					20		20
Asia	**10**	**10**	**40**	**20**	**60**	**60**	**200**
Taipei	**10**	**10**	**40**	**20**	**20**	**30**	**130**
Eslite	10	10	40	20	20	30	130
Tokyo					**40**	**30**	**70**
Tsutaya					40	30	70
Europe		**40**		**30**	**20**		**90**
Paris		**40**		**30**	**20**		**90**
Shakespeare & Company		40		30	20		90
Oceania		**60**	**40**	**50**	**30**	**20**	**200**
Sydney		**60**	**40**	**50**	**30**	**20**	**200**
MCA Store		60	40	50	30	20	200
Total	**10**	**110**	**80**	**100**	**130**	**80**	**510**

(a) The sliced snapshot at t_1.

Fig. 12. The sliced snapshots for t_1, t_3, and t_4.

	Harpercollins	Little, Brown and Company	Oxford University	Total
	Fermat's Last Theorem	The Catcher in the Rye	War and Peace	
America	**50**	**30**		**80**
Boston	**50**	**30**		**80**
Brattle Book Shop	50	30		80
Asia			**130**	**130**
Kaohsiung			**60**	**60**
Eslite			60	60
Tokyo			**70**	**70**
Kinokuniya			70	70
Oceania		**80**		**80**
Carlton		**80**		**80**
Readings		80		80
Total	**50**	**110**	**130**	**290**

(b) The sliced snapshot at t_3.

	Free Press	Oxford University	Total
	The Language of God	War and Peace	
Oceania	**15**	**25**	**40**
Sydney	**15**	**25**	**40**
Pubu	15	25	40
Total	**15**	**25**	**40**

(c) The sliced snapshot at t_4.

Fig. 12. (Cont'd) The sliced snapshots for t_1, t_3, and t_4.

4. THE FORMAL DEFINITION OF OUR FRAMEWORK

The framework of our solution has been illustrated by Fig. 1, where each record in a fact table will use the value (t_i) of the time key ($\underline{T}$) to link to the target version of a dimension table D^{t_i}. To formally illustrate the concept of *temporal data warehousing*, we define the basic elements of a temporal data warehouse in the following. Some of these elements are defined as the extended counterparts of the elements of traditional data warehouses as presented in our previous work [24]. Our previous work defines the elements of a document warehouse, which can be regarded as a multidimensional indexing structure of a document set. By combining the proposed framework and our previous work [29], a document warehouse can be extended with temporal features for the version control of each document in a document set.

4.1 Temporal Dimensional Modeling

To illustrate the proposed temporal dimensional modeling for temporal data warehouse creation, we depict a star schema in Fig. 13 with three temporal dimension tables ($D_1^{t_j}$, $D_2^{t_j}$, and $D_3^{t_j}$) linked to a fact table F, where each $D_i^{t_j}$ ($i \in \{1, 2, 3\}$) is modeled as a series of status changing relations of the same schema, with the content status respectively corresponding to t_j (*e.g.*, t_1, t_2, and t_3 in Fig. 13). Similar to traditional dimensional modeling, each primary key (PK_i) of a dimension tables $D_i^{t_j}$ is introduced into F as a foreign key FK_i ($i \in \{1, 2, 3\}$).

Periodically, records will be inserted into F with different T values (*i.e.*, the transaction times, t_1, t_2, and t_3), together with some measures (*e.g.*, m_1, m_2, …). A *measure* is an attribute of a fact table on which aggregation functions (*e.g.*, *sum*, *count*, *average*, *minimum*, *maximum*) can be applied for calculations. Based on different T values, namely t_j, the join operations of F with $D_i^{t_j}$ should be performed as the arrows indicate in Fig. 13; *i.e.*, different tuples with different t_j should be linked to the correct version of $D_i^{t_j}$ We call such operation *temporal join* and formally define it in Definition 1.

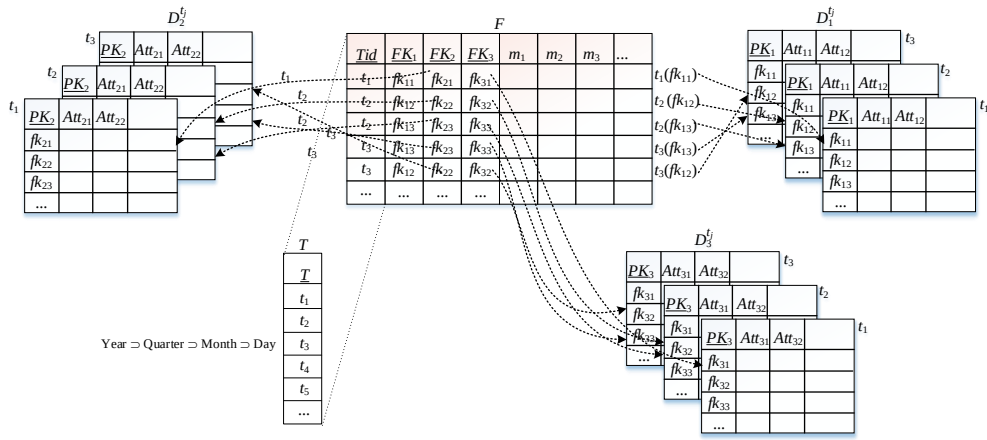

Fig. 13. The proposed temporal dimensional modeling concept.

Definition 1: A *temporal join*, denoted $F(T, K, A)⧖_K D^t(K, B)$, is a binary operation for joining a relation $F(T, K, A)$ with a temporal relation $D^t(K, B)$, where T stores time points, K is the join key and both A and B are the remaining (composite) attributes of F and D^t, respectively, such that for each $f(t, k, a)\in F$, select the tuple $d(k, b)$ contained in the version of D^t at time t, and then compose a naturally-joined tuple $r(t, k, a, b)$ as a result. To state this in a formal way by relational algebraic notations, a temporal join is defined as:

$$F(T, K, A)⧖_K D^t(K, B) = \{r(t, k, a, b) \mid (\exists f(t, k, a)\in F) \wedge (\exists d(k, b)\in D^t),$$
$$r(t, k, a, b) = \pi_{t, k, a, b}(\{f(t, k, a)\} \bowtie_{f.k=d.k} \{d(k, b)\})\}.$$

Currently, none of the mainstream commercial database management systems (DBMSs) support this temporal join operation directly. Fortunately, contemporary SQL statements, like Transact-SQL of Microsoft SQL Server, provide enough capabilities to implement this temporal join operation.

Based on Definition 1, the *temporal dimensional modeling* process can be implemented by temporally-joining a fact table $F(\underline{T}, K_1, K_2, \ldots, K_i, \ldots, K_n, A)$ with all involved temporal dimensions $D_i^t(K_i, B)$, $1 \le i \le n$:

$$(\ldots((F⧖_{K_1} D_1^{t_j} ⧖_{K_2} D_2^{t_j})\ldots⧖_{K_i} D_i^{t_j})\ldots⧖_{K_n} D_n^{t_j}).$$

Basically, the join key $F.K_i$ is a foreign key references $D_i^t.K_i$. Therefore, every tuple $(t, \ldots, k_i, \ldots, a)\in F$ is able to find a matched tuple $(k_i, b)\in D_i^t$, when performing these temporal joins. In Fig. 8, the first tuple of F inserted at t_1 should be joined with $D_1^{t_1}$, $D_2^{t_1}$, $D_3^{t_1}$ by the key values fk_{11}, fk_{21} and fk_{31}, respectively. Likewise, the second tuple of F inserted at t_2 should be linked with $D_1^{t_2}$, $D_2^{t_2}$ and $D_3^{t_2}$ by the key values fk_{12}, fk_{22} and fk_{32}, respectively. To achieve this, $F.T$ should be projected and regarded as a traditional *time* dimension T (Please notice that, since T is invariant, there is no need to create it as a temporal dimension). Then, all we need is finding a correct and efficient way to match each target tuple in the right version of each $D_i^{t_j}$ based on a time instance t_j in T through a series of temporal joins:

$(\ldots((F \natural_{K_1} D_1^{t_j}) \natural_{K_2} D_2^{t_j}) \ldots \natural_{K_i} D_i^{t_j}) \ldots \natural_{K_n} D_n^{t_j})$.

We start with defining the concept of *temporal dimension*.

Definition 2: A *temporal dimension* D^{t_j} is a multi-versioned tree structure of h levels, $h \geq 1$, with versions respectively indexed by the time instance t_j selected from a *time* dimension T. It is used to represent the hierarchical relationships among a set of keywords. A node in D^{t_j} is called a *member* and each internal node contains a special child called *summary member*, denoted '*', which is used for denoting the total concept of the other children of the internal node.

Definition 3: An *aggregated temporal dimension* D^T, is the complete data association collected and aggregated from all temporal dimensions of all versions of D^{t_j}, where t_j is an instance of the *time* dimension T.

A traditional dimension can be regarded as a special case of temporal dimension with only one version (current status). When drawing a version of a temporal dimension D^{t_j} for t, we usually leave out a summary member, since it has the same meaning with its parent node at t.

Definition 4: For a *temporal dimension* D^{t_j}, the *i^{th}-level member set* of a version, denoted $D^{t_j}(i)$, is defined as $D^{t_j}(i) = \{a \mid a$ is a member in the ith level of D^{t_j}, but a is not a summary member$\}$. We use $D^{t_j}(0)$ to denote the union of all non-summary members in D^{t_j}, which is the union of all ith level member sets in D^{t_j}. That is, $D^{t_j}(0) = \cup_{1 \leq i \leq h} D^{t_j}(i)$, where h is the height of D^{t_j}. In practice, each $D^{t_j}(i)$ has a specific name, which will be called the ith-level name. All versions of D^{t_j} use the same ith-level name; however, the members of the same level in different versions of D^{t_j} may be different.

Practically, a temporal dimension can be constructed from a temporal relation, with each level corresponding to an attribute in the relation, and these attribute names are usually used as the corresponding level names. To illustrate the above definitions, we give an example as follows.

Example 1: Suppose there is a temporal relation Ranges representing the continental regions of the world as shown in Fig. 14. This relation can be used to construct a temporal dimension, denoted R^{t_j} as depicted in Fig. 15, where the first level corresponds to the dimension itself, which is commonly denoted "(*All Ranges*)", and the second and third levels are derived from the attributes *Region* and *City*, respectively. Practically, the dimension data of all versions in Ranges should be collected and associated together by the time key ($\underline{T}$), as shown in the right-hand-side of Fig. 14, for creating the aggregated temporal dimension R^T.

All nodes in Fig. 15 with label '*' are summary members. A *summary member* groups its siblings into a set as a whole to represent the upper level of summary within a dimension hierarchy. That is, the summary member in the second level has the same meaning as *regions in the World*, which represents {*Asia*, *America*, *Europe*, *Oceania*} based on the current content. Besides, the summary members in R^{t_1} under *Asia*, *America*, *Europe* and *Oceania* have the same corresponding meaning with *Asia*, *America*, *Europe* and *Oceania*,

which denote {*Taipei*, *Tokyo*}, {*New York*}, {*Paris*} and {*Sydney*}, respectively. Likewise, the summary members in R^{t_2} under *Asia*, *America*, *Europe* and *Oceania* have the same corresponding meaning with *Asia*, *America*, *Europe* and *Oceania*, which denote {*Kaohsiung*, *Tokyo*}, {*New York*}, {*Paris*} and {*Sydney*}, respectively. Please kindly allow us to ignore the illustration of R^{t_3} for concise presentation.

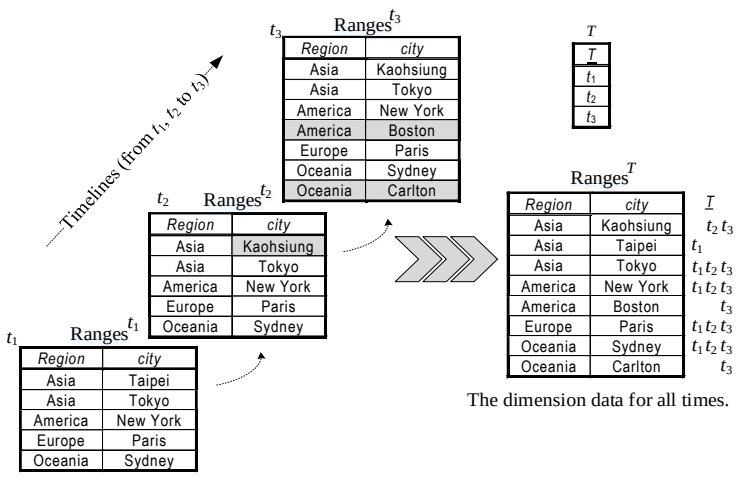

Fig. 14. A time-related relation region for constructing the *temporal dimension* R^{t_p}.

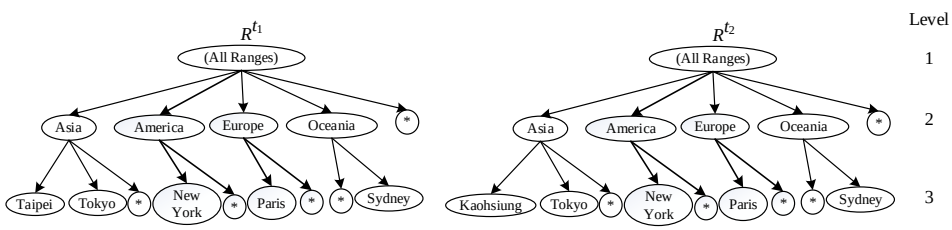

Fig. 15. An illustration of temporal dimensions R^{t_1} and R^{t_2}.

By omitting all the summary members of R^{t_p}, Fig. 15 can be concisely redrawn in Fig. 16. According to the illustration of R^{t_p}, we know that:

1. $R^{t_1}(1) = \{(All\ Ranges)\}$, $R^{t_1}(2) = \{Asia, America, Europe, Oceania\}$, and $R^{t_1}(3) = \{Taipei, Tokyo, New\ York, Paris, Sydney\}$, and $R^{t_1}(0) = \{(All\ Ranges), Asia, America, Europe, Oceania, Taipei, Tokyo, Taipei, New\ York, Paris, Sydney\}$;
2. $R^{t_2}(1) = \{(All\ Ranges)\}$, $R^{t_2}(2) = \{Asia, America, Europe, Oceania\}$, and $R^{t_2}(3) = \{Kaohsiung, Tokyo, Taipei, New\ York, Paris, Sydney\}$, and $R^{t_2}(0) = \{(All\ Ranges), Asia, America, Europe, Oceania, Kaohsiung, Tokyo, Taipei, New\ York, Paris, Sydney\}$.
3. $R^{t_3}(1) = \{(All\ Ranges)\}$, $R^{t_3}(2) = \{Asia, America, Europe, Oceania\}$, and $R^{t_3}(3) = \{Kaohsiung, Tokyo, New\ York, Boston, Paris, Sydney, Carlton\}$, and $R^{t_3}(0) = \{(All\ Ranges), Asia, America, Europe, Oceania, Kaohsiung, Tokyo, New\ York, Boston, Paris, Sydney, Carlton\}$.
4. The aggregated temporal dimensions R^T can be illustrated in Fig. 17.

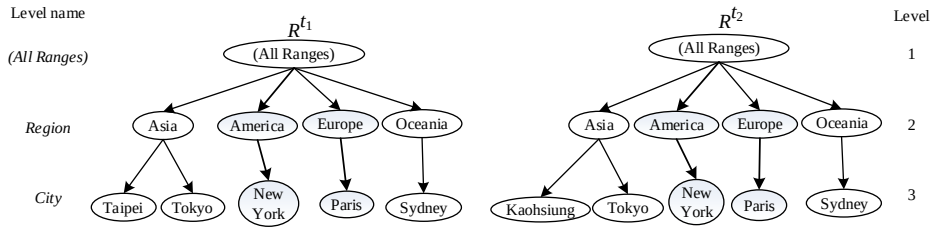

Fig. 16. A concise illustration of temporal dimensions R^{t_1} and R^{t_2}.

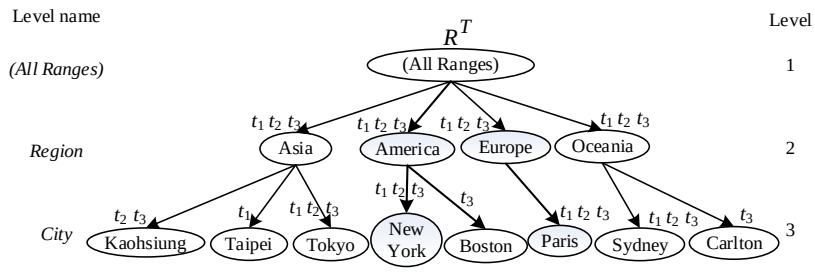

Fig. 17. The aggregated temporal dimensions R^T.

For a temporal dimension D^{t_j}, there are two basic operations called *drill-down* and *roll-up*, which are formally defined as follows.

Definition 5: For each version (for time t) of a *temporal dimension* D^{t_j}, expanding an internal node to obtain all of its children is called *drill-down at t*, and shrinking of a set of children to obtain their common parent is called *roll-up at t*.

This can be further clarified by the following definitions.

Definition 6: For any two n-tuple of keywords $A^t = (a_1, a_2, \ldots, a_i, \ldots, a_n)$ and $B^t = (b_1, b_2, \ldots, b_i, \ldots, b_n)$ defined on the versions for time t of n temporal dimensions $(D_1^{t_j}, D_2^{t_j}, \ldots, D_i^{t_j}, \ldots, D_n^{t_j})$, where a_i and $b_i \in D_i^t(0)$, we define B^t *is a member of drilling down* A^t *along* D_i^t *at t* (or A^t *is a member of rolling up* B^t *along* D_i^t *at t*), denoted $A^t \prec_i B^t$, if and only if there exists exactly an i, $1 \le i \le n$, such that b_i is a child of a_i in D_i^t, and $b_k = a_k$, for all $k \ne i$.

For an aggregated temporal dimension D^T, when users are concerning the accumulated structure of all times in T. Then, expanding an internal node to obtain all of its children is also called *drill-down*, and shrinking of a set of children to obtain their common parent is likewise called *roll-up*.

The basic component of a temporal cube is called a *t-cell*, which is defined as follows.

Definition 7: A *t-cell* $c^t = (t, C, V)$ of time t defined by

- a fact table $F(T, K_1, K_2, \ldots, K_i, \ldots, K_n, M)$,
- a *time* dimension T, projected from $F.T$,
- n aggregated temporal dimensions $D = (D_1^t, D_2^t, \ldots, D_i^T, \ldots, D_n^T)$, such that each D_i^T contains a foreign key K_i references $F.K_i$, $1 \le i \le n$,

where

- $t \in T$,
- $C = (c_1, c_2, \ldots, c_i, \ldots, c_n)$, $c_i \in D_i^t(0) \cup \{\text{'*'}\}$, $1 \le i \le n$, and
- $V = (v_1, v_2, \ldots, v_k) = (f_1(C, M_1), f_2(C, M_2), \ldots, f_k(C, M_k))$ is a tuple of values returned by respectively applying aggregate functions $f_j(C, M_j)$ on each measure in $M = \{M_1, M_2, \ldots, M_j, \ldots, M_k\}$, $1 \le j \le k$, such that the result of the temporal joins at time t $(\ldots((F \bowtie_{K_1} D_1^t) \bowtie_{K_2} D_2^t) \ldots \bowtie_{K_i} D_i^t) \ldots \bowtie_{K_n} D_n^t)$ is non-empty.

Definition 8: A *t-cell* c^t is called an m-d *t-cell* of time t, $0 \le m \le n$, if and only if there are exactly m non-summary member c_i (*i.e.*, $c_i \ne$ '*'). If $m = n$ and $c_i \in D_i^T(h_i)$, where h_i is the height of D_i^T, for all $1 \le i \le n$, then c^t is also called a *base t-cell*; otherwise c^t is called a *non-base t-cell*.

Definition 9: A *temporal cube* $C_T = (T, D, M)$, where $D = (D_1^T, D_2^T, \ldots, D_i^T, \ldots, D_n^T)$, and $M = \{M_1, M_2, \ldots, M_j, \ldots, M_k\}$, defined on a *time* dimension T, n aggregated temporal dimensions $(D_1^T, D_2^T, \ldots, D_i^T, \ldots, D_n^T)$, and k measures $\{M_1, M_2, \ldots, M_j, \ldots, M_k\}$, is an $(n+1)$-dimensional structure composed of all *t-cells* $c^t = (t, C, V)$, $c^t \in T(0) \times (\underset{1 \le i \le n}{\times} D_i^T(0))$.

Based on the above definitions, a temporal cube is a multidimensional structure with each temporal dimension can be indexed by a time point t in a traditional dimension T. It allows users to browse cells by rolling up and drilling down along some temporal dimensions, at some time intervals of different granularities. That helps us obtain further insight into time-dependent relationships.

A sample illustration of a temporal cube $C_T = (T, D, M)$, with $D = (B, R^T)$ and $M = (quantity)$, consisting of a measure of the quantity of books sold in different regions, is shown in Fig. 18, where T is a *time* dimension, B represents a traditional dimension *Books*, R^T stands for the aggregated temporal dimension of Ranges discussed in Example 1.

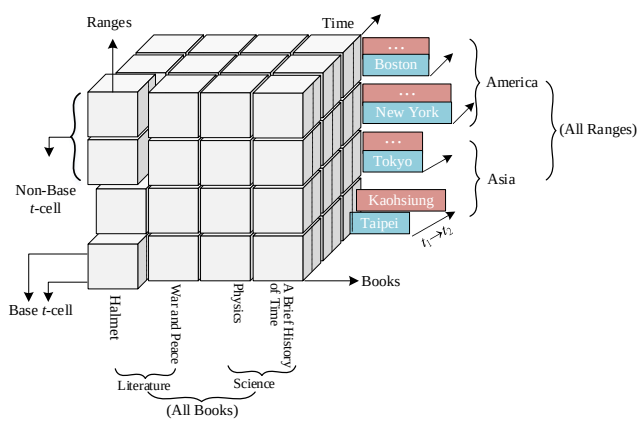

Fig. 18. A sample illustration of a temporal cube.

4.2 Spatial Dimensions

Miller & Han [17] distinguish three types of spatial dimensions for cube construction. These three types of dimension are as follows:

1. *Nongeometric spatial dimension*: contains only nominal or ordinal location of the dimension members, such as place names (*e.g.*, National Taiwan University), street addresses (University Rd.), or hierarchically structured boundaries (*e.g.*, Kaohsiung→South→Taiwan). Neither shape nor geometry nor cartographic data are used. This type of spatial dimension is fully supported by nonspatial data warehousing technology.
2. *Geometric spatial dimension*: contains a vector-based cartographic representation for every member of every level of a dimension hierarchy to allow the cartographic visualization, spatial drilling, or other spatial operation of the dimension members [1]. For instance, every city in Taiwan would be represented by a point, and every country would be represented as polygons.
3. *Mixed geometric spatial dimension*: contains a cartographic representation for some members of the dimension, and nominal/ordinal locators for the other members. A mixed spatial dimension can also contain a cartographic representation for only some members of the same hierarchy level (*e.g.*, all cities in South Taiwan, but not all cities in East Taiwan). The mixed spatial dimension offers some benefits of the geometric spatial dimension while suffering from some limitations of the nongeometric spatial dimension, all this at varying degrees depending on the type of mixes involved.

We only consider the *nongeometric spatial dimension* in our study to keep our framework neat and concise. Although this can offer only a fraction of the analytical richness of the other types of spatial dimensions [1], it is sufficient to capture digital geographic footprints for most of the cloud applications, like social networking and related analytics. When a user check-ins at some places during surfing in a social network site, the latitude and longitude can be translated into a nominal location or an approximate address (*e.g.*, by Google Map API) to help us build a spatial dimension. The slowly changing dimension problem of a spatial dimension, like city or country name changings, can be easily solved by our framework if it is built as a temporal dimension (interestingly, it is actually a spatiotemporal dimension). That is, by introducing the *nongeometric spatiotemporal dimension*, our framework is consistently applicable to both temporal and spatial dimensional modeling.

To illustrate the concept of our spatiotemporal model, we depict a graph to show the participants of a conference (namely, 2020 ICDE conference) in Fig. 19. For a concise presentation, we suppose there are totally 7 scholars. During the conference days (*i.e.*, from 2020/04/20 to 2020/04/24), each scholar chooses their convenient durations (represented in a square bracket) to attend the conference. There are two participants with slowly changing attributes (from times t_1 to t_2):

1. The city where Luna (UUID 2) was located was once called 'Taipei County' (time t_1), but it was renamed into 'New Taipei' at the conference time. Therefore, if the *city* is regarded as a temporal dimension, then 'New Taipei' should be used at t_2 for on-line analytical processing.
2. Suppose Tom (UUID 5) had a master's degree (time t_1), but when he participates in the conference (time t_2), he has already got a PhD degree.

By employing our spatiotemporal model, the following operations:

Conferences(<u>*cid*</u>, *name*, ...)$\bowtie_{cid}$ Participate(<u>*date*</u>, <u>*sid*</u>, <u>*cid*</u>) $\overline{\times}_{sid}$ Scholars^t(<u>*sid*</u>, *name*, …)

generate all the participation information as listed in Fig. 20. Then, we can correctly conduct the multidimensional analysis as Fig. 21 illustrates, where the number of participants in each conference day can be counted based on the two hierarchies: "*Affiliation* ⊃ *Degree*" and "*City* ⊃ *Gender*", respectively used for horizontal and vertical axes.

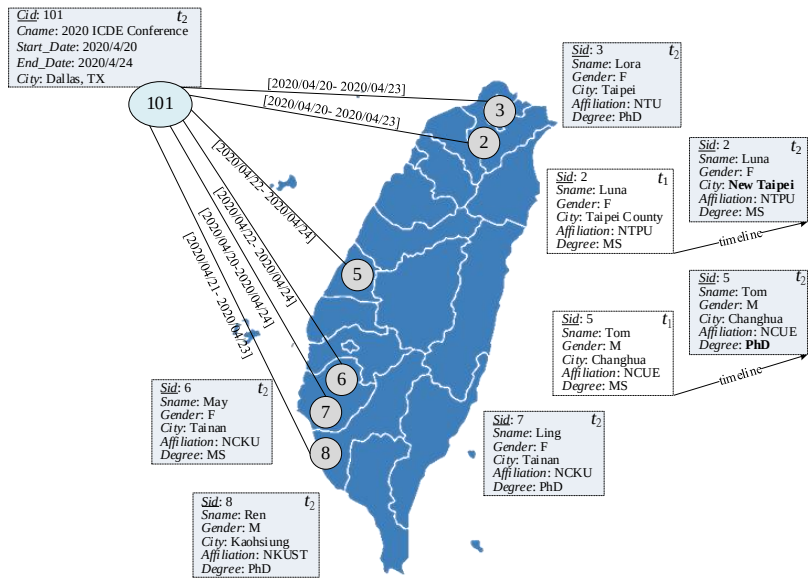

Fig. 19. A sample illustration of ICDE 2020 Conference and the participants.

sid	*sname*	*gender*	*city*	*affiliation*	*degree*	*participate_date*	*cname*
2	Luna	F	New Taipei	NTPU	MS	2020-04-20	2020 ICDE
2	Luna	F	New Taipei	NTPU	MS	2020-04-21	2020 ICDE
2	Luna	F	New Taipei	NTPU	MS	2020-04-22	2020 ICDE
2	Luna	F	New Taipei	NTPU	MS	2020-04-23	2020 ICDE
3	Lora	F	Taipei	NTU	PhD	2020-04-20	2020 ICDE
3	Lora	F	Taipei	NTU	PhD	2020-04-21	2020 ICDE
3	Lora	F	Taipei	NTU	PhD	2020-04-22	2020 ICDE
3	Lora	F	Taipei	NTU	PhD	2020-04-23	2020 ICDE
5	Tom	M	Changhua	NCUE	PhD	2020-04-22	2020 ICDE
5	Tom	M	Changhua	NCUE	PhD	2020-04-23	2020 ICDE
5	Tom	M	Changhua	NCUE	PhD	2020-04-24	2020 ICDE
6	May	F	Tainan	NCKU	MS	2020-04-22	2020 ICDE
6	May	F	Tainan	NCKU	MS	2020-04-23	2020 ICDE
6	May	F	Tainan	NCKU	MS	2020-04-24	2020 ICDE
7	Ling	F	Tainan	NCKU	PhD	2020-04-20	2020 ICDE
7	Ling	F	Tainan	NCKU	PhD	2020-04-21	2020 ICDE
7	Ling	F	Tainan	NCKU	PhD	2020-04-22	2020 ICDE
7	Ling	F	Tainan	NCKU	PhD	2020-04-23	2020 ICDE
7	Ling	F	Tainan	NCKU	PhD	2020-04-24	2020 ICDE
8	Ren	M	Kaohsiung	NKUST	PhD	2020-04-21	2020 ICDE
8	Ren	M	Kaohsiung	NKUST	PhD	2020-04-22	2020 ICDE
8	Ren	M	Kaohsiung	NKUST	PhD	2020-04-23	2020 ICDE

Fig. 20. All the participation information of 2020 ICDE.

count	New Taipei F	Tainan F	Taipei F	Total
NCKU		1		1
PhD		1		1
NTPU	1			1
MS	1			1
NTU			1	1
PhD			1	1
Total	1	1	1	3

participate_date: 2020/4/20 | 2020/4/21 | 2020/4/22 | 2020/4/23 | 2020/4/24

(a) The sliced result of the numbers of participants of 2020 ICDE (using 2020/4/20).

count	Kaohsiung M	New Taipei F	Tainan F	Taipei F	Total
NCKU			1		1
PhD			1		1
NKUST	1				1
PhD	1				1
NTPU		1			1
MS		1			1
NTU				1	1
PhD				1	1
Total	1	1	1	1	4

participate_date: 2020/4/20 | 2020/4/21 | 2020/4/22 | 2020/4/23 | 2020/4/24

(b) The sliced result of the numbers of participants of 2020 ICDE (using 2020/4/21).

	Changhua M	Kaohsiung M	New Taipei F	Tainan F	Taipei F	Total
NCKU				2		2
MS				1		1
PhD				1		1
NCUE	1					1
PhD	1					1
NKUST		1				1
PhD		1				1
NTPU				1		1
MS				1		1
NTU					1	1
PhD					1	1
Total	1	1	1	2	1	6

participate_date: 2020/4/20 | 2020/4/21 | 2020/4/22 | 2020/4/23 | 2020/4/24

(c) The sliced result of the numbers of participants of 2020 ICDE (using 2020/4/22).

	Changhua M	Kaohsiung M	New Taipei F	Tainan F	Taipei F	Total
NCKU				2		2
MS				1		1
PhD				1		1
NCUE	1					1
PhD	1					1
NKUST		1				1
PhD		1				1
NTPU				1		1
MS				1		1
NTU					1	1
PhD					1	1
Total	1	1	1	2	1	6

participate_date: 2020/4/20 | 2020/4/21 | 2020/4/22 | 2020/4/23 | 2020/4/24

(d) The sliced result of the numbers of participants of 2020 ICDE (using 2020/4/23).

count	Changhua M	Tainan F	Total
NCKU		2	2
MS		1	1
PhD		1	1
NCUE	1		1
PhD	1		1
Total	1	2	3

participate_date: 2020/4/20 | 2020/4/21 | 2020/4/22 | 2020/4/23 | 2020/4/24

(e) The sliced result of the numbers of participants of 2020 ICDE (using 2020/4/21).

Fig. 21. Multidimensional analysis of the participants of ICDE 2020 Conference.

5. THE PERFORMANCE EXPERIMENTS OF OUR FRAMEWORK

To test the feasibility of our approach, we have conducted a performance experiment by designing a BOB (including three relations: Bookstores, Orders and Books) database, where Bookstores and Books can be regarded as the temporal dimension tables, and the relation Orders be considered as the fact table. The data source of Bookstores and Books were received from the open data published in the official Website of National Central Library, Taiwan:

1. The publisher data in Taiwan were used as the data source for Bookstores, which can be found at http://isbn.ncl.edu.tw/NCL_ISBNNet/opendata/isbnpub.csv. In total, we received 963 tuples.
2. All recently published books in Taiwan were used as the data source for Books, which can be found at http://isbn.ncl.edu.tw/NCL_ISBNNet/opendata/isbn.xml. We obtained 3448 tuples after converting the XML data into flat records.

We then randomly generated orders (within the period between 1900/1/1 and 2021/12/31) into the relation Orders, intermixed with some attribute updates of the tuples in Bookstores and Books to simulate slowly changing dimensions.

To conduct the performance evaluation with different volumes, the data in Orders were partitioned into 12 sets, cumulatively every ten years. In total, there were 927,354 transactions from 1900/1/1 to 2021/12/31 in the fact table Orders. Using SQL Server 2019 as the database engine running on a PC Server of Intel(R) Xeon(R) E-2124G 4-Core CPU, 3.40GHz with 32GB memory, we tested the execution times for building the BOB cube, and the temporal dimensions Books and Bookstores, and found the performance of these primitive operations were underperforming.

To boost the performance, we tried to materialize the temporal dimensions Books and Bookstores, and then create indices on the transaction time attributes (*Tid*) of these materialized temporal dimension tables. The index creation execution time cost is extremely low and can be ignored, and the time for building the BOB cube can be extensively reduced by employing these indices (as indicated in the last two columns of Table 1). We finally illustrate the visualized performance evaluation result in Fig. 22.

The traditional slowly changing dimension processing suffers from labor-intensive and time-consuming problems for a long time. That's why R. Kimball recommends using SQL MERGE statement as a design tip in the official site[1]. However, although the MERGE statement is powerful and multifunctional, yet it can be hard to master[2]. That's why we design and define the *temporal join* operation to advocate its applications.

Even the time for building a temporal dimension seems costly, it can be achieved in parallel or incrementally. That is, by partitioning the transactions based on equal intervals, $T = T_1 \cup T_2 \ldots \cup T_i \ldots \cup T_n$, the construction of all temporal dimensions can be separately achieved on each T_i (or in parallel) using almost the same time cost. Based on the conducted performance test, we conclude that: if the temporal dimensions can be indexed by their transaction time keys and constructed incrementally, then the performance of cube construction is hopefully acceptable and feasible for industrial applications.

[1] https://www.kimballgroup.com/2008/11/design-tip-107-using-the-sql-merge-statement-for-slowly-changing-dimension-processing

[2] https://www.mssqltips.com/sqlservertip/2883/using-the-sql-server-merge-statement-to-process-type-2-slowly-changing-dimensions

Table 1. The performance evaluation of different data sets.

Data Set	*No. of transactions in Orders (in thousands)*	*Time for building the BOB cube (second)*	*Time for building Books Dimension (3448) (second)*	*Time for building Bookstores Dimension (963) (second)*	*time for indexing Books & Bookstores* ***(cost can be omitted in Fig.19)***	*Time for building the BOB cube with indices (second)*
1900~2021	927.354	1196	893	193	1	21
1910~2021	850.981	1105	847	167	2	20
1920~2021	774.978	939	700	152	1	16
1930~2021	698.691	891	623	133	2	14
1940~2021	622.377	729	536	117	2	11
1950~2021	546.538	621	470	102	0 (< 1 sec.)	10
1960~2021	470.460	495	406	89	0 (< 1 sec.)	8
1970~2021	394.852	407	329	73	0 (< 1 sec.)	7
1980~2021	319.371	333	267	60	0 (< 1 sec.)	6
1990~2021	243.560	260	209	45	0 (< 1 sec.)	4
2000~2021	167.186	182	146	32	0 (< 1 sec.)	4
2010~2021	91.526	100	77	17	0 (< 1 sec.)	3

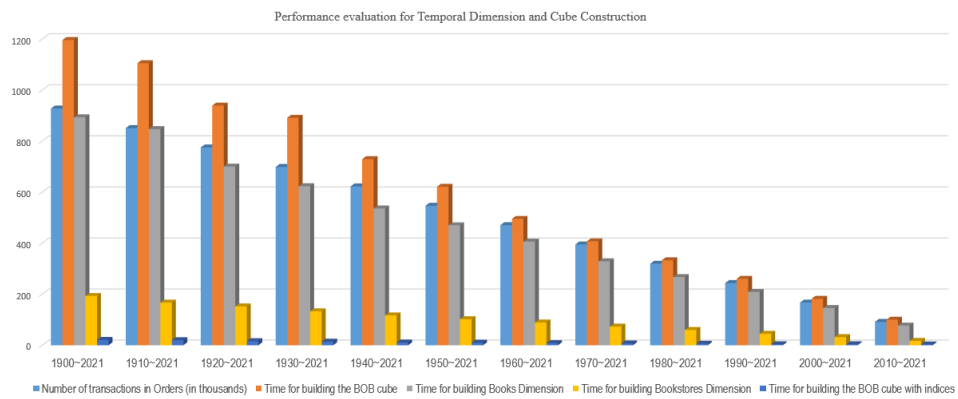

Fig. 22. The performance evaluation diagram for different data sets.

6. APPLICATIONS BASED ON OUR FRAMEWORK

We are witnessing an unprecedented growth of location-related and time-related data, which underscores the vital role of spatiotemporal processing in our society. We realize big data processing systems with spatiotemporal features, together with their data warehousing, are now underpinning many emerging data management ecosystems, in many areas of societal interest. Our spatiotemporal model can be used to model many practical applications with abstractions. In Fig. 23, we depict an IoT network consisting of different sensors, where blue vertices are used for detecting water levels in the underpasses of a mega city, and gray vertices are used to monitor hill landslides of some geolocations. Assuming their status can be divided into *normal*, *warning* and *dangerous*. When their status changes continuously along the timeline, a spatiotemporal data warehousing system can be built by deriving the *t*-cells of a temporal cube for each *t* moment, such that the number of dangerous spots can be visualized and calculated instantly for administrative decision making. If the number of dangerous spots run over a threshold, then by drilling through to target the dangerous sensors, the city government can activate the alarm system for traffic control or an emergency procedure for possible evacuation.

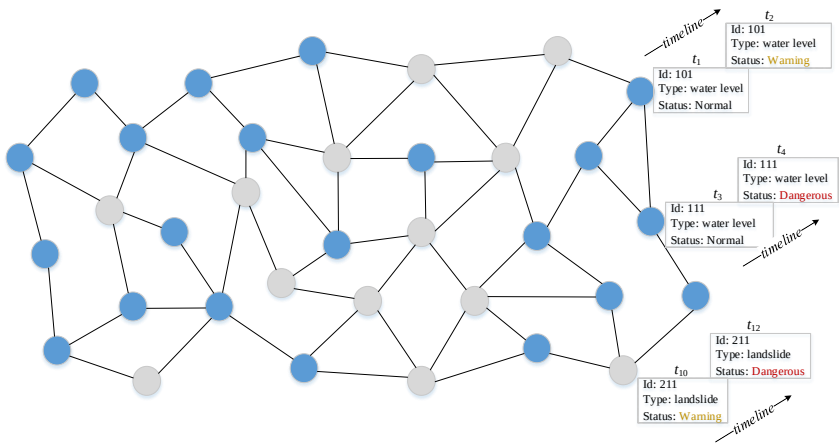

Fig. 23. An example IoT network consisting of different sensors.

By combining with the location-based service supported by mobile devices and utilizing a resource multiplexer, various multidimensional analyses for different kinds of spatiotemporal business intelligence can be conducted. For instance, to prevent the Covid-19 pandemic, each vertex in Fig. 23 can also be regarded as a branch of some chain stores. By gathering the cellphone check-in information (arriving at irregular intervals) of all customers in a branch, our framework can help enterprise administrators derive the status of each branch, to grasp the number of customers at different timestamps of each branch located at different geolocations. If a customer (with the cellphone number) is notified as suspected of being infected, then the multidimensional summarization result may effectively help administrators make a correct decision to fit the official epidemic prevention policy.

7. RESOLVING THE SCD PROBLEMS WITH EFFECTIVENESS

Based on the type definitions of SCD [13, 19], we may verify how our method resolves different types of the SCD problem with effectiveness as follows.

1. *Type 0–Retain Original*: as the dimension attribute value never changes, so the dimension *D* has only one version and facts are always grouped by this original value.
2. *Type 1–Overwrite*: the old and new attribute values in the temporal dimension can be kept to reflect the status, and therefore our approach will keep everything to be retrieved by a time index and the history is not destroyed.
3. *Type 2–Add New Row*: there is no need to add a new row in the temporal dimension with the updated attribute values, as our approach can keep every status for each changing time by the dimension modeling process.
4. *Type 3–Add New Attribute*: there is no need to add a new attribute in the temporal dimension to preserve the old attribute value, as our approach keeps every status for each value changing.
5. *Type 4–Add Mini-Dimension*: when a group of attributes in a dimension rapidly changes, all the status can be captured in all the versions of a temporal dimension, and there is no need to split off to a *mini-dimension*.

6. *Type 5–Add Mini-Dimension and Type 1 Outrigger*: the temporal dimension can be used to accurately preserve historical attribute values, and all historical facts can be linked to the correct attribute values.
7. *Type 6–Add Type 1 Attributes to Type 2 Dimension*: every temporal dimension delivers both historical and current dimension attribute values, as it keeps exactly one version for each changing time.
8. *Type 7–Dual Type 1 and Type 2 Dimensions*: we now unify the traditional hybrid technique of supporting both as-was and as-is reporting into a single temporal dimensional modeling approach.
9. *Type N–Preserving a Complete History of Changes in a Nested Relation*: Although the proposed approach needs no additional relations, columns or rows, and requires no extra join operations or surrogate keys, it still cannot be implemented in a relational database management system directly (unless additional relations and surrogate keys are introduced, which induce extra join operations are also needed). Our approach has no such problem.

8. CONCLUSION AND FUTURE DIRECTIONS

We have proposed a general framework of dimensional modeling for spatiotemporal data warehouse creation. The approach also mitigates the cumbersome problems of SCD management.

The integration of data from different sources may be object-centric or people-centric. The former identifies objects with their unique identifiers, and their engaged persons, locations, times and events. If every object has a unique identifier (UID), then this work is easy to achieve, as objects are "honest" and do not hide themselves. The latter links different personal identifiers in different networks together, and collects the resources, together with the corresponding locations, time and events, which is no trivial task, as people may deliberately use different identifiers to hide themselves in cyberspace. That is why *entity resolution* [22, 25, 28] is important in the whole process, and we believe our framework helps developers cultivate user-centered technologies for the needs of digital footprints integration and social business intelligence for various domains, if the entity resolution problem can be resolved.

Based on the experimental result, the construction of temporal dimensions costs a lot of times, which is proportional to the quantity of transactions without any index creation. However, by adding indices on the transaction time keys of materialized temporal dimensions, the cube creation can be boosted and constructed in an acceptable situation. Basically, although an index creation takes space proportional to the number of transactions and the granularity of the time key, the execution time cost is extremely low and can be omitted, which makes our approach acceptable and feasible for industrial applications.

Currently, thanks to the temporal table functionalities that comply with the ANSI SQL: 2011 standard, spatiotemporal data management and data warehousing in commercial database management systems is becoming feasible. Although we have explored the elements of a spatiotemporal data warehouse and formally established the structure, it still needs more elaboration on defining an extended query language for the implementation of

such a system, our future work will be primarily focused on defining such language features and developing feasible approaches to realize our framework. Besides, the proposed method keeps every change of status. In the future, we aim to tackle the challenge of alleviating too much information being stored, organized and processed to make our approach more feasible for IoT applications under edge computing environments.

REFERENCES

1. A. Abello and C. Martín, "The data warehouse: an object-oriented temporal database," in *Proceeding of the 8th Conference on Jornadas Ingeniería del Software y Bases de Datos*, 2003, pp. 675-684.
2. A. Albrecht and F.Y. Naumann, "Managing ETL processes," in *Proceedings of International Workshop on New Trends in Information Integration*, 2008, pp. 12-15.
3. T. Alsahfi, M. Almotairi, and R. Elmasri, "A survey on trajectory data warehouse," *Spatial Information Research*, Vol. 28, 2020, pp. 53-66.
4. B. Bebel, J. Eder, C. Kon,cilia, T. Morzy, and R. Wrembel, "Creation and management of versions in multiversion data warehouse," in *Proceedings of ACM Symposium on Applied Computing*, 2004, pp. 717-723.
5. S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Record*, Vol. 26, 1997, pp. 65-74.
6. C. J. Date, H. Darwen, and N. Lorentzos, *Time and Relational Theory: Temporal Databases in the Relational Model and SQL*, The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, San Francisco, CA, 2014.
7. B. Devlin, "Managing time in the data warehouse," *InfoDB*, Vol. 11, 1997, pp. 7-12.
8. D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, Chapter 5, Cambridge University Press, UK, 2010.
9. G. Garani, G. K. Adam, and D. Ventzas, "Temporal data warehouse logical modelling," *International Journal on Data Mining, Modelling and Management*, Vol. 8, 2016, pp. 144-159.
10. G. Garani, N. Cassavia, and I. K. Savvas. "An application of an intelligent data warehouse for modelling spatiotemporal objects," *International Journal of Big Data Intelligence and Applications*, Vol. 1, 2020, pp. 36-57.
11. G. Garani and G. K. Adam, "A semantic trajectory data warehouse for improving nursing productivity," *Health Information Science and Systems*, Vol. 8, 2020, Article: 25.
12. M. Golfarelli and S. Rizzi, "A survey on temporal data warehousing," *International Journal of Data Warehousing and Mining*, Vol. 5, 2009, pp. 1-17.
13. W. H. Inmon, *Building the Data warehouse*, 4th ed., Wiley & Sons, Indianapolis, IN, 2005.
14. R. Kimball and J. Caseta, *The Data Warehouse ETL Toolkit – Pratical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, Wiley Publishing, IN, 2004.
15. R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed., John Wiley & Son, Indianapolis, IN, 2013.
16. K. Kulkarni and J.-E. Michels, "Temporal features in SQL: 2011," *ACM SIGMOD Record*, Vol. 41, 2012, pp. 34-43.

17. H. J. Miller and J. Han, *Geographic Data Mining and Knowledge Discovery*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, CRC Press, FL, 2009.
18. G. Ozsoyoglu and R. Snodgrass, "Temporal and real-time databases: A survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, 1995, pp. 513-532.
19. V. Santos and O. Belo, "No need to type slowly changing dimensions," in *Proceedings of IADIS Information Systems Conference*, 2011, pp. 129-136.
20. A. Simitsis, P. Vassiliadis, and T. Sellis, "Extraction-transformation-loading processes," *Encyclopedia of Database Technologies and Applications*, L. C. Rivero, J. H. Doorn and V. E. Ferraggine (eds.), Idea Group, Hershey, PA, 2005.
21. R. T. Snodgrass, *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann Publishers, CA, 2000.
22. J. R. Talburt, *Entity Resolution and Information Quality*, Morgan Kaufmann Publisher, CA, 2012.
23. W. Tan, M. B. Blake, I. Saleh, and S. Dustdar, "Social-network-sourced big data analytics," *IEEE Internet Computing*, Vol. 17, 2013, pp. 62-69.
24. F. S. C. Tseng and A. Y. H. Chou, "The concept of document warehousing for multidimensional modeling of textual-based business intelligence," *Decision Support Systems*, Vol. 42, 2006, pp. 727-744.
25. F. S. C. Tseng and A. Y. H. Chou, "Spatiotemporal multidimensional modeling of data warehouse for event tracing applications," in *Proceedings of International Computer Symposium*, 2020, Paper No. 143.
26. A. Vaisman and E. Zimanyi, "What is spatiotemporal data warehousing," in *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, LNCS Vol. 5691, 2009, pp. 9-23.
27. K. A. Vidhya, R. Soorya, N. Saranavan, T. V. Geetha, and M. Singaravelan, "Entity resolution for symptom vs disease for top-K treatments," in *Proceedings of the 4th IEEE International Conference on Advanced Computing and Communication Systems*, 2017, pp. 1-8.
28. J. Wang, T. Kraska, M. J. Franklin and J. Feng, "CrowdER: crowdsourcing entity resolution," in *Proceedings of the VLDB Endowment*, Vol. 5, 2012, pp. 1483-1494.
29. F. Zemke, "What's new in SQL: 2011," *ACM SIGMOD Record*, Vol. 41, 2012, pp. 67-73.

Frank S. C. Tseng (曾守正) received the BS, MS and Ph.D. degrees, all in Computer Science and Information Engineering from National Chiao Tung University, Taiwan, in 1986, 1988, and 1992, respectively. Dr. Tseng is one of the winners of Acer Long Term Ph.D. dissertation prize in 1992. He joined the faculty of the Department of Information Management, Yuan-Ze University, Chung-Li, Taiwan on August, 1995. From 1996 to 1997, he was the Chairman of the Department of Information Management. Then, he joined the faculty of the Department of Information Management, National Kaohsiung University of Science and Technology, Taiwan on August 1997. He was the Chairman of the department from 2009 to 2011, and selected as a Distinguished Professor from 2014 to 2019. His current research interests include distributed database systems, data warehousing and data mining, social network analytics, cloud and edge computing, service science and design thinking applications. Dr. Tseng currently is an associate editor of *Decision Support Systems* Journal. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Annie Y. H. Chou (周韻寰) received her BS degree in Applied Mathematics, M.S. degree in Computer Science and Information Engineering, and Ph.D. degree in Computer and Information Science, all from National Chiao Tung University, Taiwan, in 1987, 1989, and 1996, respectively. From 1989 to 1992, she was an Assistant Researcher of Chunghua Telecom Laboratories, Taiwan. She joined the faculty of the Department of Computer and Information Science, ROC Military Academy, in August 1997. Her research interests include mathematical analysis of computer algorithms, social networking applications, data warehousing and data mining, and cloud computing. Professor Chou was a member of the Phi Tau Phi Scholastic Honor Society.