# Slowly Changing Dimension Handling in Data Warehouses Using Temporal Database Features

Thanapol Phungtua-Eng[1] and Suphamit Chittayasothorn[2](✉)

[1] Faculty of Business Administration and Information Technology, Rajamangala University of Technology Tawan-Ok, Chakrabongse Bhuvanarth Campus, Bangkok 10400, Thailand
thanapol.phu@cpc.ac.th

[2] Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
suphamit.ch@kmitl.ac.th

**Abstract.** This paper presents the use of temporal database features to solve the Slowly Changing Dimension (SCD) problem of data warehouses. The SCD problem is presented and existing solutions, together with their limitations are shown. Temporal database features of SQL are described. Temporal data retrieval and temporal data manipulations, together with illustrated examples are demonstrated. The solution to the SCD problem is shown with illustrated examples. The data warehouse whose dimension tables are validtime state tables, but the fact table is a conventional fact table without any timestamp or validtime period, is proposed. The identifier integrity of dimension instances is preserved. The sample fact table, dimension tables, and the SQL codes which perform temporal operations to solve the problem are presented. The proposed solution gives correct results regardless of the number of changes made to the attribute of the dimension table, thus completely solves the Slowly Changing Dimension problem.

**Keywords:** Slowly Changing Dimension · Temporal data warehouse · Temporal SQL

## 1 Introduction

A data warehouse is a centralized data source for data analytics and management information making supports. Typical data warehouses comprises facts and dimensions.

Data warehouses help ease the data integration issues in organizations that employ individual special purpose information systems, which are separately developed. Many organizations deploy such information systems from many vendors or software developing teams without any prior agreement on data formats and value standards.

For many other organizations, which may not afford the costly centralized database, or have to deploy ready-to-use information systems from several vendors due to the deployment time constraint, building data warehouses is the alternative solution. Data

https://doi.org/10.1007/978-3-030-14799-0_58

warehouses employ star schemas or snowflake schemas, which comprise fact and dimension table schemas. Dimension table schemas are schemas that describe object types, which are of particular interest to the organization. Each row of the dimension table represents an object instance, which is uniquely identified by a surrogate key. The use of surrogate keys ease the join operations between the dimension tables and the fact tables. A fact table schema comprises surrogate keys from the associated dimension tables and at least one attribute called the measure. Typically, there is a time dimension. The smallest time granularity is normally at the day level since a data warehouse mostly keep summarized data as opposed to detailed transactions. A data warehouse fact table can also be represented as a cube. Each axis represent a dimension and the value at a coordinate is a measure value.

## 2 The Slowly Changing Dimensions Problem

Naturally, attribute values of existing rows of dimension tables may be updated. This could affect the analytic results. The changes made to an attribute value of a dimension, in principle, should not affect the past information of the data warehouse. However, if not handled properly, the changes could yield incorrect analytic results from the data warehouse. This dimensional attribute's value change problem is widely known in the data warehouse area as the Slowly Changing Dimensions (SCD) problem [1, 2]. The term "slowly" reflects that the values are updated occasionally and infrequently.

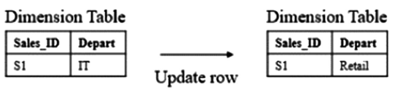
A classic example is the case where a sale person changes his team. The person used to belong to a sale team but later moved to a new team. The team name of this person is therefore changed at a particular point of time. An ideal data warehouse should be able to show that the person's sale amount before the change belongs to the former team, and the sale amount after the change belongs to the new team. This problem sounds easy to handle but it is not. Current solutions have limitations.

Solutions have been proposed and referred to as Type 1, Type 2 and Type 3 solutions. They are approaches based on the conventional relational database. All of them has some limitations. In this paper, we propose an approach to solve the SCD problem using the temporal relational database, which solves the SCD problem and all the limitations of the existing solutions.

Kimball [1] a pioneer of data warehousing, and Jensen et al. [2], classify three typical approaches for the SCD problem handling. They are summarized as follow:

### 2.1 Type 1 (Overwrite)

The type one approach simply overwrite the value of the changed attribute. The new value replaces the old value. This is a simple approach to the problem. It does not solve the problem but it is simple and easy to implement. All related facts, which are referred to by the dimension instance now refer to the new value. Figure 1 illustrates this approach. The sale person S1 moves from IT department to Retail department. This approach simply replaces IT by Retail. All the fact instances of S1 are now refer to the new department Retail even though they are old facts.



Dimension Table

| Sales_ID | Depart |
|---|---|
| S1 | IT |

Update row

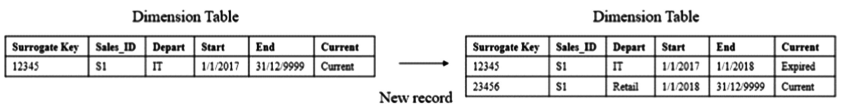
Dimension Table

| Sales_ID | Depart |
|---|---|
| S1 | Retail |

**Fig. 1.** The overwrite approach to the SCD problem

## 2.2 Type 2 (Add New Dimension Instance)

The type two approach is an attempt to preserve the history. It creates a new row in the dimension table for the dimension instance with the newly updated value. The newly created row with the new column value also has a new surrogate key. The dimension instance therefor has a new row and a new surrogate key for each change. This simple approach works but lacks the identification integrity. Each object instance should have a unique identifier; otherwise, the two instances are interpreted as different object instances. Figure 2 demonstrates the type two approach.



Dimension Table

| Surrogate Key | Sales_ID | Depart | Start | End | Current |
|---|---|---|---|---|---|
| 12345 | S1 | IT | 1/1/2017 | 31/12/9999 | Current |

New record

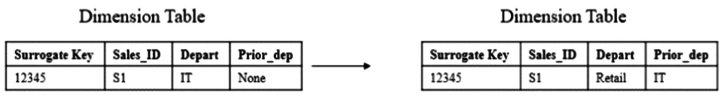
Dimension Table

| Surrogate Key | Sales_ID | Depart | Start | End | Current |
|---|---|---|---|---|---|
| 12345 | S1 | IT | 1/1/2017 | 1/1/2018 | Expired |
| 23456 | S1 | Retail | 1/1/2018 | 31/12/9999 | Current |

**Fig. 2.** The ‘add new dimension instance’ approach to the SCD problem

## 2.3 Type 3 (Add a New Attribute for Each Version)

The Type 3 approach add a new attribute to the dimension instance row. Each instance has only one row and one unique identifier. The identification integrity is preserved. However, this approach can handle only limited number of changes. Figure 3 demonstrates the Type 3 approach. Here only two versions of the department value can be accommodated.



Dimension Table

| Surrogate Key | Sales_ID | Depart | Prior_dep |
|---|---|---|---|
| 12345 | S1 | IT | None |

Dimension Table

| Surrogate Key | Sales_ID | Depart | Prior_dep |
|---|---|---|---|
| 12345 | S1 | Retail | IT |

**Fig. 3.** The ‘add new attribute’ approach to the SCD problem

There are other attempts to solve the SCD problems. Snapshot tables [3] are used to represent a dimension instance row at a point of time. The separation of a historical table from the current table is also suggested [4]. However, the instances from the two tables use different surrogate keys.

Surrogate key-based temporal data warehouse [5], another interesting approach, is an attempt to use the validtime concept of temporal database to show the validity of the changed value. However, different surrogate keys are still used for different version of the changed dimension instance. This simple approach works but still lacks the identification integrity.

Versioning tables [6, 7] are extensions to the surrogate key-based temporal data warehouse approach. A current flag is introduced to ease programming tasks and can be viewed according to applications' requirements. Different surrogate keys are still used for different versions of the changed dimension instance.

In this paper, we present an approach to the Slowly Changing Dimension problem. This approach completely solves the problem without any previous limitations. We do not replace the old value by the new value and lost track of the old one likes the Type 1 approach. We do not use different surrogate keys to refer to the same object instance likes the Type 2 approach. This identification integrity problem of the Type 2 approach obviously lead to the inability to track information of the same object instance over different periods since different identifiers are used to identify the same dimensional object instance. We do not have the limitation on number of versions of the changes that the Type 3 solution has.

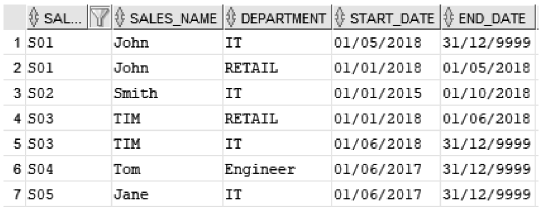
Our approach does not use different surrogate key value for different changed versions. The unique identifier of each dimension instance remains unchanged. This is feasible due to the deployment of temporal database technology in modern relational database systems and the temporal features of the SQL language as described in the following section.

## 3 Temporal Database Operations

The handling of the SCD problem without the well-known limitations, and still maintain the identification integrity so that tuples which refer to the same object instance, but have different attribute values due to the changes, and still use the same identifier is a big challenge in the data warehouse area. In this paper, we propose the use of temporal database technology to handle the problems. In order to understand our proposed solution, temporal database operations must be well understood first. This section gives a brief summary of temporal database operations in SQL.

In conventional database systems, only facts that are currently true are stored. In reality, facts in the databases change from time to time. Previously true facts are also considered important and might be referred to by applications. The database should contain not only the current information of an employee but also previous versions of the information as well. Validtime denotes the time when a fact is true in reality. Relational database tables that support validtime is called validtime state tables. Nowadays several commercially available relational database systems support validtime state tables and their SQL languages support temporal aspects of validtime state tables.

Validtime state tables keep time varying facts and the time that the fact is valid. For each time varying fact type, there are validtime start and validtime end, in the closed-open format. Each validtime interval is closed at its lower bound, and open at its upper bound to facilitate the continuity checking. Facts which are currently valid, have the upper bound valid, to-date, set to either null or 'infinity', which is the maximum date 31/12/9999. Removed facts, are not actually deleted but instead are marked as invalid facts. Figure 4 shows the validtime state table used to present the sale person dimension of a data warehouse. The validtime is in the close-open format. The two rows of each sale person with sales_id S1 and S3 are the rows before and after the changes of their department. The previous department value was RETAIL and the current one is IT. A new row is inserted for each modification. The row with the old value is marked as valid up to the day the update took place and a new row with the new value is inserted. The end date of the new row is set to 31/12/9999 to show that the row is the current one.



| | SAL... | SALES_NAME | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|---|
| 1 | S01 | John | IT | 01/05/2018 | 31/12/9999 |
| 2 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 |
| 3 | S02 | Smith | IT | 01/01/2015 | 01/10/2018 |
| 4 | S03 | TIM | RETAIL | 01/01/2018 | 01/06/2018 |
| 5 | S03 | TIM | IT | 01/06/2018 | 31/12/9999 |
| 6 | S04 | Tom | Engineer | 01/06/2017 | 31/12/9999 |
| 7 | S05 | Jane | IT | 01/06/2017 | 31/12/9999 |

**Fig. 4.** A validtime state table SALES_DIMENSION shows sale persons change departments

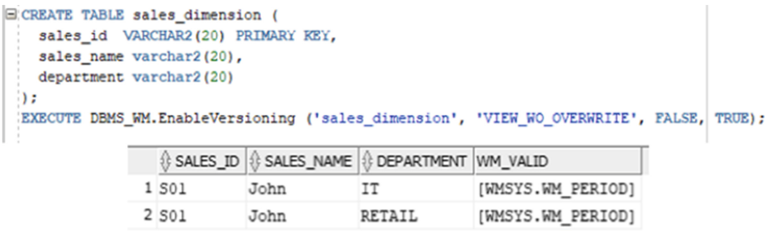
We use the Oracle Workspace Manager [8] to demonstrate relevant temporal database operations. The temporal features in this section are used to implement our approach to solve the SCD problem.

## 3.1 Validtime State Table Creation

The concept of validtime state table enables temporal queries; as well as temporal insert, delete and update operations. These temporal database operations are tedious to implement using conventional relational database technology [9]. The validtime period, if handled by conventional relational systems, needs long SQL codes to manipulate and queries. Instead, if the validtime period is handled by the temporal feature of the supported DBMS, the SQL codes are much more concise. The following commands as shown in Fig. 11 create the validtime state table sales_dimension which is to be used as our temporal dimension table for the SCD problem handling. The EXECUTE DBMS_WM.EnableVersioning command enables validtime versioning.

The attribute WM_VALID is automatically created and handled by the DBMS. It comprises the validtime attributes (WM_VALID.VALIDFROM) and (WM_VALID. VALIDTILL).

For each temporal operation, a validtime period, the period of applicability, needs to be specified. The EXECUTE DBMS_WM.SetValidTime (START_DATE, END_DATE) command therefore needs to be issued before each temporal operation. In the Fig. 5, sales_id is the apparent primary key of the validtime state table. The actual primary key, according to the conventional relational database model, is the combination of the apparent primary key sales_id and the validtime period WM_VALID. This actual primary key is handled by the temporal feature of the DBMS (the Oracle Workspace Manager).



```
CREATE TABLE sales_dimension (
  sales_id  VARCHAR2(20) PRIMARY KEY,
  sales_name varchar2(20),
  department varchar2(20)
);
EXECUTE DBMS_WM.EnableVersioning ('sales_dimension', 'VIEW_WO_OVERWRITE', FALSE, TRUE);
```

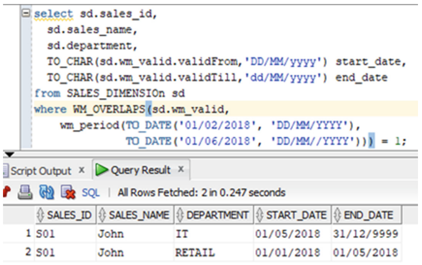
|  | SALES_ID | SALES_NAME | DEPARTMENT | WM_VALID |
|---|---|---|---|---|
| 1 | S01 | John | IT | [WMSYS.WM_PERIOD] |
| 2 | S01 | John | RETAIL | [WMSYS.WM_PERIOD] |

**Fig. 5.** The SQL command for validtime state table creation and the table with some rows

## 3.2 Temporal Query Features

Since our proposed solution to the SCD problem needs the temporal query features to handle the problem, dimension tables are created as validtime state tables. Temporal queries on them are now possible. Temporal operators comparable to the Allen's Interval Algebra [10] are available. Oracle's WM_OVERLAPS is Alllen's P1 overlaps P2. WM_CONTAINS is P1 contains P2. WM_MEETS is P1 Meets P2. WM_EQUALS is P1 equals P2.

Figure 6 demonstrates the WM_OVERLAPS operation. The SQL select statement retrieves rows of the validtime state table SALES_DIMENSION from Fig. 5 where the valid period overlaps 01/02/2018 and 31/05/2018 (the close-open format excludes 01/06/2018). The result shows two rows which belong to the sale person S01. The first row is his current row where he belongs to the IT department since 01/05/2018 until now (31/12/9999). The second row shows his previous appointment at the RETAIL department between 01/01/2018 and 30/04/2018 (01/05/2018).

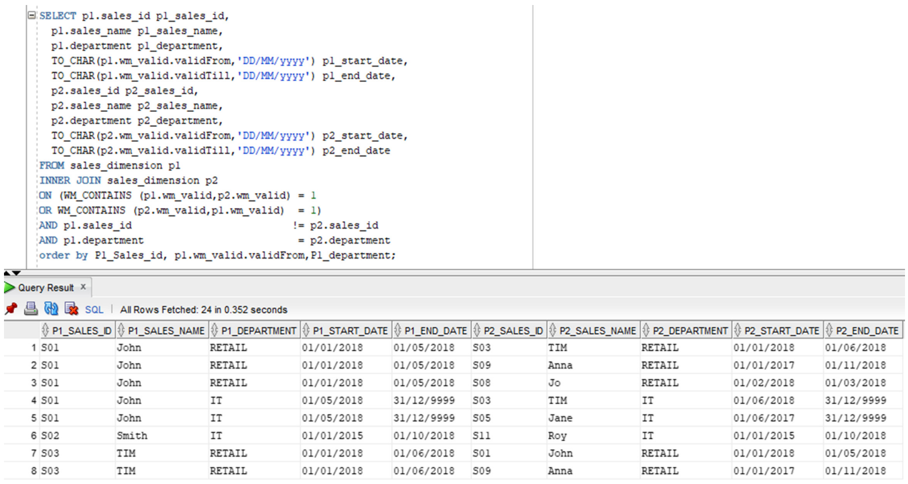


```
select sd.sales_id,
  sd.sales_name,
  sd.department,
  TO_CHAR(sd.wm_valid.validFrom,'DD/MM/yyyy') start_date,
  TO_CHAR(sd.wm_valid.validTill,'dd/MM/yyyy') end_date
from SALES_DIMENSIOn sd
where WM_OVERLAPS(sd.wm_valid,
    wm_period(TO_DATE('01/02/2018', 'DD/MM/YYYY'),
              TO_DATE('01/06/2018', 'DD/MM//YYYY'))) = 1;
```

|  | SALES_ID | SALES_NAME | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|---|
| 1 | S01 | John | IT | 01/05/2018 | 31/12/9999 |
| 2 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 |

**Fig. 6.** An SQL command that demonstrates the WM_OVERLAPS operation

Figure 7 demonstrates the join of the SALES_DIMENSION validtime state table to itself with the WM_CONTAINS condition. The query gives a list of sale persons who work during the same period of time for the same department. Temporal join operations are required to join the dimension tables with valid time and the fact tables of our proposed temporal data warehouse.



```
SELECT p1.sales_id p1_sales_id,
  p1.sales_name p1_sales_name,
  p1.department p1_department,
  TO_CHAR(p1.wm_valid.validFrom,'DD/MM/yyyy') p1_start_date,
  TO_CHAR(p1.wm_valid.validTill,'DD/MM/yyyy') p1_end_date,
  p2.sales_id p2_sales_id,
  p2.sales_name p2_sales_name,
  p2.department p2_department,
  TO_CHAR(p2.wm_valid.validFrom,'DD/MM/yyyy') p2_start_date,
  TO_CHAR(p2.wm_valid.validTill,'DD/MM/yyyy') p2_end_date
FROM sales_dimension p1
INNER JOIN sales_dimension p2
ON (WM_CONTAINS (p1.wm_valid,p2.wm_valid) = 1
OR WM_CONTAINS (p2.wm_valid,p1.wm_valid)  = 1)
AND p1.sales_id                          != p2.sales_id
AND p1.department                         = p2.department
order by P1_Sales_id, p1.wm_valid.validFrom,P1_department;
```

Query Result

SQL | All Rows Fetched: 24 in 0.352 seconds

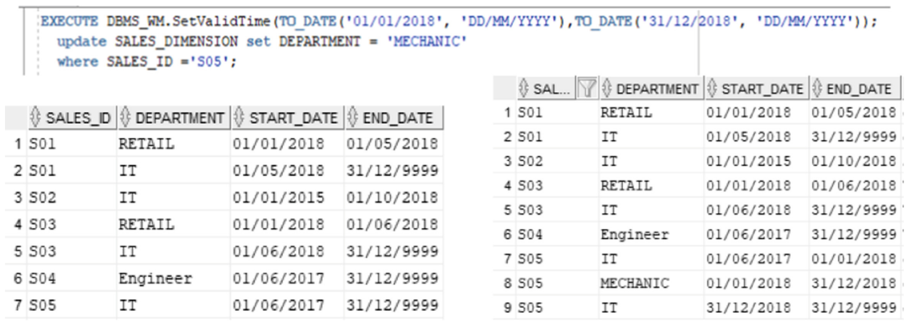
| | P1_SALES_ID | P1_SALES_NAME | P1_DEPARTMENT | P1_START_DATE | P1_END_DATE | P2_SALES_ID | P2_SALES_NAME | P2_DEPARTMENT | P2_START_DATE | P2_END_DATE |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 | S03 | TIM | RETAIL | 01/01/2018 | 01/06/2018 |
| 2 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 | S09 | Anna | RETAIL | 01/01/2017 | 01/11/2018 |
| 3 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 | S08 | Jo | RETAIL | 01/02/2018 | 01/03/2018 |
| 4 | S01 | John | IT | 01/05/2018 | 31/12/9999 | S03 | TIM | IT | 01/06/2018 | 31/12/9999 |
| 5 | S01 | John | IT | 01/05/2018 | 31/12/9999 | S05 | Jane | IT | 01/06/2017 | 31/12/9999 |
| 6 | S02 | Smith | IT | 01/01/2015 | 01/10/2018 | S11 | Roy | IT | 01/01/2015 | 01/10/2018 |
| 7 | S03 | TIM | RETAIL | 01/01/2018 | 01/06/2018 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 |
| 8 | S03 | TIM | RETAIL | 01/01/2018 | 01/06/2018 | S09 | Anna | RETAIL | 01/01/2017 | 01/11/2018 |

**Fig. 7.** An SQL query which demonstrates the join of the SALES_DIMENSION validtime state table to itself with the WM_CONTAINS condition

## 3.3 Temporal Data Manipulations

Insertion of new rows to the validtime state table is a straight forward operation. A period of validity needs to be declared before the insert operation. Update and delete operations on validtime state tables, however, are complicated and require careful considerations [9]. Period of applicability needs to be declared and considered when update or delete operations are issued. It is then checked against the period of validity of existing data. There could be several overlap patterns between the two periods. The temporal database feature of the DBMS handles them automatically. In this section, the update of attribute value is demonstrated. It will be used to modify attributes of dimension tables which are validtime state tables, thus leads to the solving of the SCD problem.

The commands in Fig. 8 demonstrate the department change of the sale person S05 to the MECHANIC department from 01/01/2018 to 30/12/2018. Note the close-open format, 31/12/2018 is not included in the period. The left table shows the sale



```
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01/01/2018', 'DD/MM/YYYY'),TO_DATE('31/12/2018', 'DD/MM/YYYY'));
  update SALES_DIMENSION set DEPARTMENT = 'MECHANIC'
  where SALES_ID ='S05';
```

| | SALES_ID | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|
| 1 | S01 | RETAIL | 01/01/2018 | 01/05/2018 |
| 2 | S01 | IT | 01/05/2018 | 31/12/9999 |
| 3 | S02 | IT | 01/01/2015 | 01/10/2018 |
| 4 | S03 | RETAIL | 01/01/2018 | 01/06/2018 |
| 5 | S03 | IT | 01/06/2018 | 31/12/9999 |
| 6 | S04 | Engineer | 01/06/2017 | 31/12/9999 |
| 7 | S05 | IT | 01/06/2017 | 31/12/9999 |

| | SAL... | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|
| 1 | S01 | RETAIL | 01/01/2018 | 01/05/2018 |
| 2 | S01 | IT | 01/05/2018 | 31/12/9999 |
| 3 | S02 | IT | 01/01/2015 | 01/10/2018 |
| 4 | S03 | RETAIL | 01/01/2018 | 01/06/2018 |
| 5 | S03 | IT | 01/06/2018 | 31/12/9999 |
| 6 | S04 | Engineer | 01/06/2017 | 31/12/9999 |
| 7 | S05 | IT | 01/06/2017 | 01/01/2018 |
| 8 | S05 | MECHANIC | 01/01/2018 | 31/12/2018 |
| 9 | S05 | IT | 31/12/2018 | 31/12/9999 |

**Fig. 8.** The temporal update operation that moves S05 to the MECHANIC department for a year

dimension table before the update. S05 was in the IT department since 01/01/2017. The update result on the right table shows three rows of S05. The END_DATE of row number 8 is updated to 01/01/2018 and two new rows are inserted.

## 4 Data Warehouses with Temporal Dimensions

Based on the temporal database technology, we propose the data warehouse with temporal dimensions. The dimension tables whose attributes' values can be modified and referred to when queried, are validtime state tables. SQL queries on them only consider the apparent primary keys. Temporal operators can be employed. The fact table is a conventional fact table without the validtime.

Figure 9 shows the star schema whose dimension SALES_DIMENSION is a validtime state table to accommodate the department changes. SALES_ID is the sale person key. It is the apparent primary key, which is unique for each dimension instance. Unlike other related works in the literatures, the primary key value will not be changed after updates on the temporal attribute. The data warehouse that employs such validtime dimension tables, can now be referred to as a temporal data warehouse. Note that there are no such validtime periods in dimension tables of other conventional data warehouses.

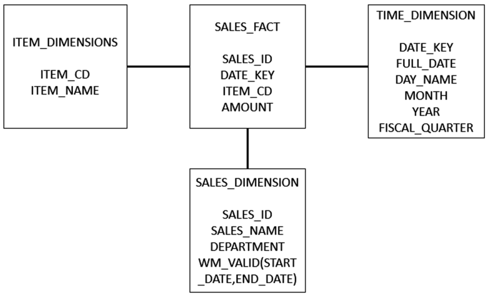
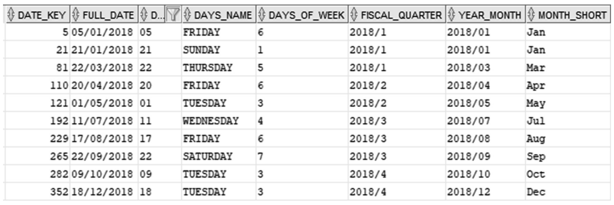


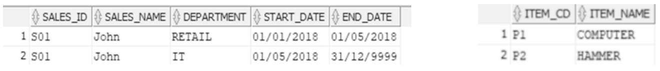
**Fig. 9.** A star schema whose dimension SALES_DIMENSION is a validtime state table

Figure 10 shows the time dimension table TIME_DIM. Note that the surrogate key DATE_KEY is used instead of the actual date. This is a common practice in data warehousing. The attribute FULL_DATE is the actual date. When the SCD problem is detected, this date will be used to check validity of the corresponding fact instances. Figure 11 shows the sale persons dimension SALES_DIM, which is a validtime state table. The item dimension table ITEM_DIM is a conventional dimension table. Figure 12 is the fact table SALES_FACT together with its fact instances of S01. The fact table does not have a validtime but it is possible to include one here. Note also the use of surrogate keys for easy references.



| DATE_KEY | FULL_DATE | D.. | DAYS_NAME | DAYS_OF_WEEK | FISCAL_QUARTER | YEAR_MONTH | MONTH_SHORT |
|---|---|---|---|---|---|---|---|
| 5 | 05/01/2018 | 05 | FRIDAY | 6 | 2018/1 | 2018/01 | Jan |
| 21 | 21/01/2018 | 21 | SUNDAY | 1 | 2018/1 | 2018/01 | Jan |
| 81 | 22/03/2018 | 22 | THURSDAY | 5 | 2018/1 | 2018/03 | Mar |
| 110 | 20/04/2018 | 20 | FRIDAY | 6 | 2018/2 | 2018/04 | Apr |
| 121 | 01/05/2018 | 01 | TUESDAY | 3 | 2018/2 | 2018/05 | May |
| 192 | 11/07/2018 | 11 | WEDNESDAY | 4 | 2018/3 | 2018/07 | Jul |
| 229 | 17/08/2018 | 17 | FRIDAY | 6 | 2018/3 | 2018/08 | Aug |
| 265 | 22/09/2018 | 22 | SATURDAY | 7 | 2018/3 | 2018/09 | Sep |
| 282 | 09/10/2018 | 09 | TUESDAY | 3 | 2018/4 | 2018/10 | Oct |
| 352 | 18/12/2018 | 18 | TUESDAY | 3 | 2018/4 | 2018/12 | Dec |

**Fig. 10.** The Time dimension table TIME_DIM



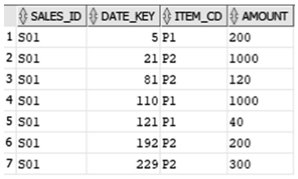
| | SALES_ID | SALES_NAME | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|---|
| 1 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 |
| 2 | S01 | John | IT | 01/05/2018 | 31/12/9999 |

SALES_DIM

| | ITEM_CD | ITEM_NAME |
|---|---|---|
| 1 | P1 | COMPUTER |
| 2 | P2 | HAMMER |

ITEM_DIM

**Fig. 11.** Rows of S01 from the dimension tables SALES_DIM, and rows of ITEM_DIM



| | SALES_ID | DATE_KEY | ITEM_CD | AMOUNT |
|---|---|---|---|---|
| 1 | S01 | 5 | P1 | 200 |
| 2 | S01 | 21 | P2 | 1000 |
| 3 | S01 | 81 | P2 | 120 |
| 4 | S01 | 110 | P1 | 1000 |
| 5 | S01 | 121 | P1 | 40 |
| 6 | S01 | 192 | P2 | 200 |
| 7 | S01 | 229 | P2 | 300 |

**Fig. 12.** Rows of the fact table SALES_FACT which show fact instances of S01

## 5 Slowly Changing Dimension Handling

In this section, we present the use of temporal database features to solve the Slowly Changing Dimension (SCD) problem. We propose that the dimension tables, whose attributes are significant enough so that their changes are to be tracked and shown as output attributes of the data warehouse, be validtime state tables. The surrogate keys of such dimension tables are the apparent primary keys. Each entity instance has the surrogate primary key, which does not changed with time. Attribute value updates and the validtime, are handled by the temporal feature of the DBMS as described in the previous section.

The key idea is to relate each row of the fact table to the correct row of the dimension table whose attribute values have been changed. In our approach, since the apparent primary key remains the same regardless of the number of changes made to an attribute value, and the number of attribute value versions, simple joining of common surrogate keys of the fact table and the corresponding dimension table is not enough. The join must be a temporal join with time overlap or contain checking.

The next issue is the time validity of the fact instance rows of the fact table. In principle, a row of a fact table refers to the time key of the time dimension. This time key represents a point of time, not a period of time, and is a surrogate key. The SCD dimension table, on the other hand, has a validtime period. A possible representation is to attach a time period for the day to each fact instance row, thus complicate the Extract

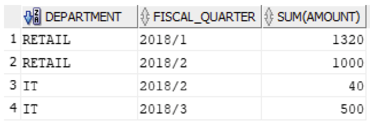
Transfer Load (ETL) task of the data warehouse. This validtime on the fact table would also lead to the wrong interpretation that each fact instance is only valid for the day, which is not true. We therefore keep the fact table as a conventional relational database table.

To check that the date on the fact instance row corresponds with the validtime period of a corresponding dimension row, we extract the full date of the fact instance row by using a join with the time dimension table via the time surrogate key. Once the full date of the fact instance is obtained, it can be checked if the date and the date plus one (due to the close-open format) contains in a validtime period of a dimension instance row with the same dimension surrogate key.

An illustrated example in Fig. 11 shows that the sale person S01 John was with the RETAIL department from 01/01/2018 to 30/04/2018. He then moved to the IT department from 01/01/2018 until now. Figure 10 shows some rows of the time dimension table TIME_DIM. The surrogate key, DATE_KEY, is used instead of the actual date timestamp. This is a typical data warehouse identification. Date keys 5, 21, 81, 110 represent the days when the sale person S01 was with the RETAIL department. The rest of the date keys, 121 to 352 are the days when S01 belongs to the IT department.

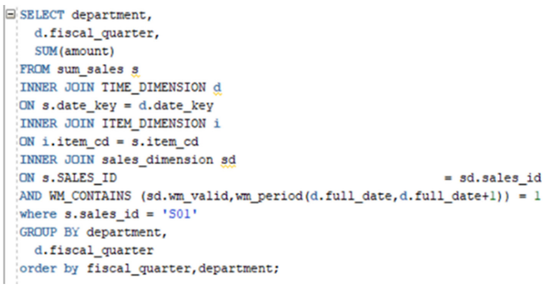
The fact table SALE_FACT in Fig. 12 shows sale fact instances some of which belong to S01 when he was with RETAIL and some when he was with IT. The first four rows of the fact table shows the sale records on the days 5, 21, 81, and 110, which are the days when S01 was with RETAIL. The other rows show the sale records on the days 121, 192, and 229 when he was already moved to IT. Notice that all these sales fact instances refer to the person as S01. The facts that he belongs to different departments during different periods of time are in the dimension table SALES_DIM which is a validtime state table.

The Slowly Changing Dimension (SCD) problem, according to the above sample data, is the case when the supplier S01 changed his department and the system cannot keep track of the changes, thus gives incorrect total values of his sales amount for each department. Figure 13 shows the correct result obtained from our data warehouse with temporal dimensions. The total sales amount of S01 when he was with the RETAIL department is 2,320 units. The first fiscal quarter of 2018 is 1,320 units and the second quarter of 2018 is 1,000 units. The total sales amount of S01 when he was with the IT department is 540 units. The first fiscal quarter of 2018 is 40 units and the second quarter of 2018 is 500 units. The fiscal quarter information is obtained from the time dimension table TIME_DIM. The SQL query, which employs the WM_CONTAINS to check fact instance and dimension instance periods of validity, is shown in Fig. 14.



| | DEPARTMENT | FISCAL_QUARTER | SUM(AMOUNT) |
|---|---|---|---|
| 1 | RETAIL | 2018/1 | 1320 |
| 2 | RETAIL | 2018/2 | 1000 |
| 3 | IT | 2018/2 | 40 |
| 4 | IT | 2018/3 | 500 |

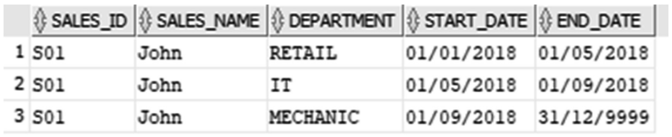
**Fig. 13.** The query result demonstrates that the SCD problem is solved.



```
SELECT department,
  d.fiscal_quarter,
  SUM(amount)
FROM sum_sales s
INNER JOIN TIME_DIMENSION d
ON s.date_key = d.date_key
INNER JOIN ITEM_DIMENSION i
ON i.item_cd = s.item_cd
INNER JOIN sales_dimension sd
ON s.SALES_ID                                    = sd.sales_id
AND WM_CONTAINS (sd.wm_valid,wm_period(d.full_date,d.full_date+1)) = 1
where s.sales_id = 'S01'
GROUP BY department,
  d.fiscal_quarter
order by fiscal_quarter,department;
```

**Fig. 14.** The SQL query which produces correct result despite the change in the dimension.

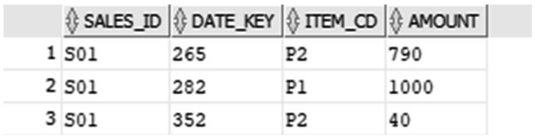
To illustrate the consequence of a further update to the attribute, suppose the sale person S01 changes his department once again. This time he moved from IT to MECHANIC department since 01/09/2018. The temporal SQL update statement marks the END_DATE of the IT row to be 01/09/2018. The sale persons dimension table SALES_DIM now has a new row as shown in Fig. 15.



| | SALES_ID | SALES_NAME | DEPARTMENT | START_DATE | END_DATE |
|---|---|---|---|---|---|
| 1 | S01 | John | RETAIL | 01/01/2018 | 01/05/2018 |
| 2 | S01 | John | IT | 01/05/2018 | 01/09/2018 |
| 3 | S01 | John | MECHANIC | 01/09/2018 | 31/12/9999 |

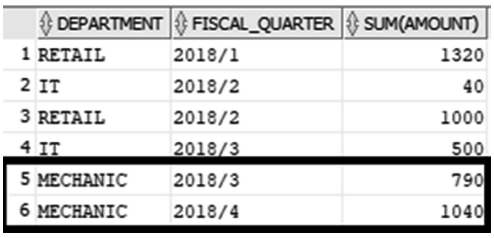
**Fig. 15.** Rows of S01 from the dimension table SALES_DIM after S01 moved to MECHANIC

The sale person makes more sales and corresponding new fact instance rows are shown in the fact table SALES_FACT as shown in Fig. 16. The date 265, 282, 352 are actually 22/09/2018, 09/10/2018, and 18/12/2018 respectively. By applying the SQL query with temporal feature from Fig. 14, the query result which shows sales by department and fiscal quarter of S01 is correctly shown in Fig. 17.



| | SALES_ID | DATE_KEY | ITEM_CD | AMOUNT |
|---|---|---|---|---|
| 1 | S01 | 265 | P2 | 790 |
| 2 | S01 | 282 | P1 | 1000 |
| 3 | S01 | 352 | P2 | 40 |

**Fig. 16.** New fact rows of S01 after moving to MECHANIC department



| | DEPARTMENT | FISCAL_QUARTER | SUM(AMOUNT) |
|---|---|---|---|
| 1 | RETAIL | 2018/1 | 1320 |
| 2 | IT | 2018/2 | 40 |
| 3 | RETAIL | 2018/2 | 1000 |
| 4 | IT | 2018/3 | 500 |
| 5 | MECHANIC | 2018/3 | 790 |
| 6 | MECHANIC | 2018/4 | 1040 |

**Fig. 17.** The query result, which shows correct sales by department and fiscal quarter of S01

## 6 Conclusions

This paper presents the use of temporal database technology to solve the Slowly Changing Dimension problem. A temporal data warehouse, which has validtime dimension tables is proposed. The identifier integrity of dimension instances is preserved. The same surrogate key identifier is used to identify the same object instance even after a dimension attribute value is changed. The dimension attribute value can be changed several times without any limitations. The query results are always correct regardless of the number of changes made, thus completely solves the Slowly Changing Dimension problem. All limitations of the previously well-known solutions of the problem are lifted. Our implementation is based on the Oracle Workspace Manager. However, relational DBMSs that adhere to the SQL standards since 2011 have the temporal SQL features as presented in the paper and therefore can be used to implement the proposed solution.

## References

1. Kimbal, R., Ross, M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 3rd edn. Wiley, Indianapolis (2013)
2. Jensen, C., Pederson, T.B., Thomsen, C.: Multidimensional Databases and Data Warehousing. Lexington, Morgan Claypool (2010)
3. Nguyen, T.M., Tjoa, A.M., Nemec, J., Windisch, M.: An approach towards an event-fed solution for slowly changing dimensions in data warehouses with a detailed case study. Data Knowl. Eng. **63**(1), 26–43 (2007)
4. Santos, V., Belo, O.: No need to type slowly changing dimensions. In: Proceedings of IADIS International Conference Information Systems 2011, Avila, Spain, pp. 11–13 (2011)
5. Faisal, S., Sarwar, M.: Handling slowly changing dimensions in data warehouses. J. Syst. Softw. **94**, 151–160 (2014)
6. Ravat, F., Teste, O., Zurfluh, G.: A multiversion-based multidimensional model. In: Tjoa, A. M., Trujillo, J. (eds.) DaWaK 2006. LNCS, vol. 4081, pp. 65–74. Springer, Heidelberg (2006). https://doi.org/10.1007/11823728_7
7. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses. In: Wang, S., et al. (eds.) ER 2004. LNCS, vol. 3289, pp. 415–428. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30466-1_38

8. Workspace Manager Valid Time Support. https://docs.oracle.com/database/121/ADWSM/long_vt.htm
9. Snodgrass, R.T.: Managing temporal data – a five part series, Database programming and design, TimeCenter technical report (1998)
10. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**, 832–843 (1983)