MODERN TEMPORAL DATA MODELS: PROPERTIES AND DEFICIENCIES

Dušan Petković

University of Applied Sciences
Rosenheim, 83024, Germany
petkovic@fh-rosenheim.de

Abstract

The time is generally a challenging task. All issues in relation to time can be better supported using temporal data models. More than two-dozen such models have been introduced in time period between 1988 and 1995. After this period, the work on temporal data has been not so dynamic. The last couple of years brought the new revival of the topic and the emergence of new data models. In this article we present the final release of temporal data model, which is published as the part of the latest SQL standard. After that we discuss the differences between the specification and one of its implementations. Finally, we conclude the article with discussion of properties of this data model and advocate for introduction of the PERIOD data type in one of the future versions of the specification.

Keywords - temporal databases, valid time, transaction time, bitemporal, standardization, SQL:2011

1 INTRODUCTION

After many years of hard work, the ISO standardization committee for SQL has released the final specification for temporal data. This specification has its roots in several proposals, which came from different sources. One of the most important differences to the previous proposals is that the specification does not build an entirely new part, as planned before.

The story of temporal data specification for the SQL standard is very long and rich on twists. In the year 1994, the ANSI department working on the SQL standard had started the work on temporal data. (The American National Standards Institute - ANSI - is the organization that oversees the development of standards in the United States.) They made a proposal, which was based upon the work of Richard Snodgrass and his colleagues. Professor Snodgrass has previously published a specification of temporal language, which was an extension of the SQL standard at that time [6]. The language specification, together with other materials has been published later in a book [7].The American proposal has not been accepted by the ISO committee due to several significant insufficiencies [1]. (The International Standard Organization, ISO, is the international counterpart to ANSI.) At the same time, the members of the English standardization committee made another proposal, which was based upon the work of Nikos Lorentzos [3].

As a reaction to the reject of the American proposal, the members of the ANSI committee did not agree with the proposal of their British colleagues, hence none of these specifications has been accepted at that time. For this reason, the next SQL standard, SQL:1999, [4] did not contain the specification of temporal data at all. In the following years, there were no attempts to solve this problem and to make a specification for temporal data. There was a deadlock between members of the ISO committee, and hence, in the year 2001 the SQL standardization committee decided to abandon the work on temporal data.

In the year 2007, the members of SQL standardization committee started the work concerning system-versioned tables. At the same time they made the decision to add the already existing specification to SQL/Foundations. (The last SQL standard comprises nine parts, which are not consecutively numbered. The most important part is the second one, SQL/Foundations, which comprises the foundations of the language. For this reason, this part is the most voluminous of all existing parts.)

In the year 2010, the committee started the work concerning application-time period tables. The both extensions, application-time period tables and system-versioned tables, build the biggest part of the new specification for temporal data, which has been released in the SQL:2011 standard. Generally, the new specification inherits a lot of ideas from the both previous proposals, has however a significantly different syntax. (A list of all temporal extensions in SQL:2011 can be found in [2], while all non-temporal extensions are described in [9].)

This article presents the temporal data model proposed by the SQL standardization committee. We also discuss the differences between this specification and the implementation of temporal data in IBM DB2. The main contribution of this paper is to show deficiencies of the proposed model and to present the PERIOD data type, which should be considered in future specifications.

The rest of this article is organized in the following way: Section 2 deals with the syntax extensions in relation to application-time period tables. The new syntax of CREATE TABLE e.g. ALTER TABLE statement is given, as well as the syntax extensions in INSERT, DELETE and UPDATE statements. Section 3 describes system-versioned tables. Section 4 discusses a specification of a new table form (called bitemporal table), where application-time period data as well as system-versioned data are combined. Section 5 shows the implementation of system-versioned, application-time period and bitemporal tables in IBM DB2. The last section gives conclusions and discusses future work.

2 APPLICATION-TIME PERIOD TABLES

An application-time period table is a table that contains a PERIOD clause with a user-defined name for time period. The SQL:2011 standard restricts such a table so that its rows are associated with one or more temporal periods. A typical example of such a problem is an insurance application, where it is necessary to keep track of insurance information (art of insurance, annual premium etc.) of a given customer that are in effect at any given point in time.

An application-time period table contains two additional columns, one to store the start time of a period associated with the row and one to store the end time. Values of both columns are set by the user. Additional syntax is provided for users to specify primary key/unique constraints to ensure that no two rows with the same key value have overlapping periods.

Note that application-time period tables are related to a temporal dimension called valid time. Valid time concerns the time when an event is true in the real world. For this reason, this form of time is independent of its storage in a database and can concern the past, present and future snapshots of the event. Using timestamps, it is possible to form different versions of an event. This is the central aspect of a temporal database realization.

2.1 Creating Application-Time Period Tables

When creating an application-time period table, two additional columns must be defined. The former stores start values and the latter the end values of the corresponding time period. The both columns must be NOT NULL and their data type can be either DATE or TIMESTAMP. The interval specified by the values of these columns is half open, meaning that it contains the value of the start column but not the value of the end column. The both columns are specified in the CREATE TABLE or ALTER TABLE statement, using the PERIOD clause. This clauses specify names of both columns explicitly and the implicit rule that start_date<end_date. (The name of this time period is specified by the user.) Example 1 shows the creation of an application-time period table.

Example 1

```
CREATE TABLE a_employees (emp_idVARCHAR(30) NOT NULL,
dept_name VARCHAR (20) NOT NULL,
dept_id VARCHAR(30),
start DATE NOT NULL,
end DATE NOT NULL,
   PERIOD FOR emp_period (start, end),
PRIMARY KEY (emp_id, emp_period WITHOUT OVERLAPS),
   FOREIGN KEY (dept_id, PERIOD emp_period)
       REFERENCES department (dept_id, PERIOD dept_period));
```

The example above shows two other extensions of the CREATE TABLE statement in relation to application-time period tables: The first one is the WITHOUT OVERLAPS clause. This clause forbids overlapping of time periods for the same value of non-temporal part of the primary key. Additionally, the specification of the PERIOD clause in the FOREIGN KEY option forbids the existence of a row in a referencing table whose time period is not contained in the time period of a corresponding referenced table.

2.2 Retrieving and Modifying Data from Application-Time Period Tables

The syntax of the INSERT statement for application-time period tables is identical to the syntax of the same statement for convenient tables. This means that the start and end time of the period has to be explicitly specified by the user. (The both values can be related to the past, present or future.) Example 2 shows the insertion of a row in the **a_employees** table, while Table 1 displays the table's content after insertion.

Example 2

```
INSERT INTO a_employees (emp_id, dept_name, dept_id, start, end)
   VALUES ('e1', 'Marketing', 'd1', DATE'2010-01-15', DATE '2011-01-15');
```

emp_id	dept_name	dept_id	start	end
e1	Marketing	d1	2010.01.15	2011.01.15

Table 1: The content of the **a_employees** table

As we already stated, the WITH OVERLAPS clause forbids overlapping of time periods for the same value of the non-temporal part of the primary key. For this reason, the INSERT statement in Example 3 will produce an error.

Example 3

```
INSERT INTO a_employees (emp_id, dept_name, dept_id, start, end)
   VALUES ('e1', 'Marketing', 'd1', DATE'2010-04-01', DATE '2010-12-31');
```

The insertion of a row into the**a_employees** table in Example 3 will not be executed, because of the existence of the **emp_period**integrity rule in the PRIMARY KEY clause in Example 1. The time period of the row in Example 3 ('2010-04-01', '2010-12-31') overlaps the time period of the already inserted row ('2010-01-15', '2011-01-15').

The syntax of the UPDATE statement is extended with the FOR PORTION clause to support temporal data. This clause is used to specify the time period for which the modification in the UPDATE statement is applied. Example 4 shows the use of this clause and Table 2 displays the content of the table after modification.

Example 4

```
UPDATE a_employees FOR PORTION OF emp_period
FROM DATE '2010-05-01' TO DATE '2010-08-01'
     SET dept_id = 'd2'  WHERE emp_id = 'e1';
```

emp_id	dept_name	dept_id	Start	End
e1	Marketing	d1	2010.01.15	2010.05.01
e1	Marketing	d2	2010.05.01	2010.08.01
e1	Marketing	d1	2010.08.01	2011.01.15

Table 2: The content of the **a_employees** table after modification

The time period specified in the FOR PORTION clause in Example 4 divides the time period of the already inserted row in two parts. For this reason after the execution of the UPDATE statement the table will contain two new inserted rows and the previous row, with the modified time period

The DELETE statement can be used with its convenient syntax or with the same extension as the UPDATE statement. In Example 5, the FOR PORTION clause specifies the time period, for which the deletion is applied.

Example 5

```
DELETE FROM a_employees FOR PORTION OF emp_period
FROM DATE '2010-06-01' TO DATE '2011-01-01';
```

emp_id	dept_name	dept_id	Start	End
e1	Marketing	d1	2010.01.15	2010.05.01
e1	Marketing	d2	2010.05.01	2010.06.01
e1	Marketing	d1	2011.01.01	2011.01.15

Table 3: The result of the DELETE statement in Example 5

The DELETE statement in Example 5 concerns the time periods of the second and third row in Table 2: For this reason, these two time periods will be “shortened” according to the specified period('2010-06-01', '2011-01-01'). Table 3 shows the content of the table after execution of the DELETE statement.

3 SYSTEM-VERSIONED TABLES

System-versioned tables are intended to solve real world problems, where the history of data modifications must be maintained. The structure of system-versioned tables is extended with two new columns that contains begin and the end of the specified time period. The values of these columns contain system times, which are updated each time the table content is modified.

System-versioned tables are related to a temporal dimension called transaction time. Transaction time concerns the time the fact was present in the database as stored data. In other words, the transaction time of an event describes the times, where the event is stored in a database and presents the correct image of the modelled world. Timestamps of transaction time events are defined according to the schedule adopted by the operating system. Therefore, we can build the history of all such timestamps in relation to the past and current time, but not in relation to future. For this reason, system-versioned tables contain system times, which are updated each time the table content is modified.

3.1 Creating System-Versioned Tables

The names of the two new columns described above are specified by the user, but their values are inserted by the system. The syntax of the CREATE TABLE statement contains several new extensions, which can be seen in Example 6.

Example 6

```
CREATE TABLE s_employees(emp_nameVARCHAR(50) NOT NULL,
dept_id VARCHAR(10),
system_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
system_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME (system_start, system_end),
PRIMARY KEY (emp_name))
WITH SYSTEM VERSIONING;
```

The new clauses in CREATE TABLE statement in relation to system-versioned tables are:

- GENERATED ALWAYS AS ROW START
- GENERATED ALWAYS AS ROW END

The former clause specifies begin of the time period, while the latter defines the end of that period. Therefore, the column **system_start** in Example 6 stores values in relation to begin of the system time period and the column **system_end**the values of the end of it. The PERIOD FOR SYSTEM TIME clause contains the names of both columns and the given order of them implies the rule that the value of the first column must be always earlier than that of the second one. The last option, WITH SYSTEM VERSIONING, inserts implicitly the start time values to the corresponding values of the column, which build the primary key. The reason for this is that in a case of system-versioned tables, the values of a non-temporal column, which builds the primary key, are not unique, because several versions of such a column can exist. An instance of temporal entity must have a primary key composed of time-varying and non-time-varying attributes. Therefore, the values of activation start time are used as the part of the primary key.

The most important attitude of system-versioned tables is that old versions of an instance are preserved. In spite of it, the current instance (one which contains the current time) and those with previous (historical) times are treated differently: The former is called current system row and only that one will be modified with update operations. The old versions of the same table are called historical rows and they are read-only. Note, that already specified constraints are valid only for the current row(s).

3.2 Retrieving and Modifying Data from System-Versioned Tables

The syntax of the INSERT statement for system-versioned tables is identical to the syntax of the same statement for convenient tables. The values of both system time columns are implicitly inserted by the system. If a new row is inserted in the system-versioned table, the current time is assigned to the column with the start time, while the highest possibly timestamp is inserted into the column with the end time. Example 7 uses the INSERT statement to add a new row to the **s_employees** table, while Table 4 displays the content of the table after insertion. Note that although the time-variant columns of the **s_employees** table are defined using the TIMESTAMP data type, only the DATE portion of them will be shown in results, because of simplicity. (The assumed current time for this example is 2012.08.01.)

Example 7

```
INSERT INTO s_employees (emp_name, dept_id)

VALUES ('Scott', 'd1');
```

dept_id	emp_name	system_start	system_end
d1	Scott	2012.08.01	9999.12.31

Table 4: The content of the **s_employees** table after insertion

Concerning the UPDATE statement, the columns with the start and end time cannot be used explicitly in the SET clause of that statement. When a row of the system-versioned table is modified, the old version of that row is preserved, before the column values are modified and the current version is inserted. At the same time, the end time of the old version and the begin time of the new one will be set to the current (transaction) time. (The DELETE statement has the same semantics as the UPDATE statement.)

The modification of the **s_employees** table is shown in Example 8, while Table 5 displays the content of that table after update. Note that the “deleted” rows still belong to the content of the table. Only their activation end time will be set to the current time. (The current of execution the UPDATE statement time is 2012.08.08.)

Example 8

```
UPDATE s_employees

  SET dept_id = 'd2'
```

WHERE emp_name = 'Scott' ;

dept_id	emp_name	system_start	system_end
d1	Scott	2012.08.01	2012.08.08
d2	Scott	2012.08.08	9999.12.31

Table 5: The content of the **employee** table after update

The syntax of the SELECT statement in relation to system-versioned tables is the same as for the regular tables. The only difference is the necessity to retrieve old versions of rows. This can be done using the FOR SYSTEM_TIME clause. The meaning of this clause is to deliver rows, which satisfy the given condition. The resulting rows can be current or old version rows, depending on the condition. There are four different forms of this clause:

- FOR SYSTEM_TIME AS OF CURRENT TIMESTAMP
- FOR SYSTEM_TIME AS OF <datetime value expression>
- FOR SYSTEM_TIME BETWEEN < date value expression 1> AND< date value expression 2>
- FOR SYSTEM_TIME FROM < date value expression 1> TO< date value expression 2>

The first form of the clause is the default value. This means that if a query includes any explicit form of the FOR SYSTEM_TIME clause, this form is implicitly assumed and the query returns the current rows as the result. The second form of the clause is used to retrieve rows of a table at a specified point in time. In contrast to the second form of the FOR SYSTEM_TIME clause, the third and the fourth form specify the condition as a time period. (The former defines a closed interval, while the latter specifies a half open interval.) Examples 9 and 10 show second and the third form of the FOR SYSTEM_TIME clause, respectively.

Example 9

```
SELECT dept_name
    FROM s_employees FOR SYSTEM_TIME AS OF DATE '2010-01-01'
WHERE emp_name = ´Scott´;
```

Example 10

```
SELECT dept_name
FROM s_employees FOR SYSTEM_TIME BETWEEN DATE '2010-01-01' AND DATE '2012-01-01'
WHERE emp_name = 'Scott';
```

4 BITEMPORAL TABLES

A bitemporal table comprises both an application-time period table as well as a system-versioned table. To understand why the “marriage” of both table forms is useful in the real world, let us take a look at an example. During their existence, departments of a firm can change their names. Typically, the modification of a department name happens at a specific time, but that name is changed in the database not at the same time (usually later). In that case, the system-time period automatically records when a particular name is inserted into the database and the application time period records when the name was actually modified.

4.1 Creating Bitemporal Tables

When creating a bitemporal table, four additional temporal columns must be specified, two concerning system times and two in relation to application-time period. Example 11 shows the creation of such a table. (The CREATE TABLE statement in Example 11 is just a union of columns and clauses from Example 1 and Example 6.)

Example11

```
CREATE TABLE bi_employees (emp_idVARCHAR(30) NOT NULL,
dept_name VARCHAR (20) NOT NULL,
dept_id VARCHAR(30),
start DATE NOT NULL,
end DATE NOT NULL,
system_startDATE NOT NULL GENERATED ALWAYS AS ROW START,
system_end DATE NOT NULL GENERATED ALWAYS AS ROW END,
   PERIOD FOR SYSTEM_TIME (system_start, system_end),
PERIOD FOR emp_period (start, end),
PRIMARY KEY (emp_id, emp_period WITHOUT OVERLAPS))
)   WITH SYSTEM VERSIONING;
```

Because bitemporal tables combine properties of both forms of temporal tables, all DML statements (SELECT, INSERT, UPDATE and DELETE) can be used either for the application-time period, system-time period or the combination or both. In other words, there are no syntactic extensions which are specific for bitemporal tables.

5 IMPLEMENTATION OF TEMPORAL DATA IN IBM DB2

Several vendors of RDBMSs have already implemented temporal data. Some of them used the prerelease of the specification for implementation, while others used proprietary syntax and semantics. At this moment, there is only one vendor, who implemented the specification of temporal data from the SQL standard: IBM DB2. For this reason, we will describe the support of temporal data in IBM DB2 in this section.

5.1 Tables with Business Time

The implementation of application-time period tables in DB2 is similar to the corresponding specification in SQL:2011: The main difference is in the terminology: In DB2, such tables are called tables with business time [5].

A. Creating Tables with Business Time

The syntax of the CREATE TABLE statement for creation of tables with business time is slightly different to the corresponding syntax for creation of application-time period tables. Example 12 shows how the CREATE TABLE statement can be used to create tables with business time.

Example 12

```
CREATE TABLE a_employees (emp_idVARCHAR(30) NOT NULL,
dept_name VARCHAR (20) NOT NULL,
dept_id VARCHAR(30),
start DATE NOT NULL,  end DATE NOT NULL,
   PERIOD BUSINESS_TIME (start, end),
   PRIMARY KEY (emp_id, BUSINESS_TIME WITHOUT OVERLAPS));
```

The only difference to the corresponding SQL:2011 specification is that IBM DB2 does not allow users to define names for the specified time period (in the PERIOD clause). This is replaced by the BUSINESS_TIME reserved keyword.

B. Retrieving and Modifying Data from Tables with Business Time

The syntax of the INSERT statement in DB2 is identical to the syntax of the same statement in the SQL:2011 specification, while the syntax of the UPDATE and DELETE statements in DB2 is slightly different: The FOR PORTION clause contains the BUSINESS_TIME reserved word, instead of the user-defined name. (This is an implication from the definition of the PERIOD clause in the CREATE TABLE statement in DB2.) Example 13 shows the use of the UPDATE statement to modify tables with business time, while deletion of rows is given in Example 14. (These two examples correspond to Examples 4 and 5. For this reason, the result of these two statements is given in Table 2 and Table 3, respectively.)

Example 13

```
UPDATE a_employees FOR PORTION OF BUSINESS_TIME
     FROM DATE '2010-05-01' TO DATE '2010-08-01' SET dept_id = 'd2'    WHERE emp_id = 'e1';
```

Example 14

```
DELETE FROM a_employees FOR PORTION OF BUSINESS_TIME
           FROM DATE '2010-06-01' TO DATE '2011-01-01';
```

5.2 Tables with System Times

The semantics of system-versioned tables in DB2 is different than the semantics of the corresponding specification in SQL:2011. Instead of one system-versioning table, DB2 supports two tables, one to store current rows and one to store old versions of them. Besides that, the terminology is different: System-versioning tables are called tables with system time.

There are three steps in defining tables with system time:

- Create the base table (for current rows)
- Create the versioning table (for old versions of rows)
- Alter the base table to enable versioning and identify the versioning table

A. Creating Tables with System Time

The syntax of the CREATE TABLE statement for creating base tables with business time is almost identical to the corresponding syntax for creation of system-versioning tables. Example 15 shows how the CREATE TABLE statement can be used to create a base table with system time.

Example 15

```
CREATE TABLE s_employees (emp_nameVARCHAR(50) NOT NULL,
dept_id VARCHAR(10),
system_start TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
system_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
trans_start TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START,
            PERIOD SYSTEM_TIME (system_start, system_end),
             PRIMARY KEY (emp_name));
```

The only difference to the corresponding syntax in the SQL:2011 specification is the existence of an additional column (in our example **trans_start**), which stores transaction start time. IBM DB2 uses the values stored in this column to track when the transaction first executed a statement that modifies the content of the table. Examples 16 and 17 show the second and third step in defining tables with system time.

Example 16

```
CREATE TABLE v_employees LIKE s_employees;
```

Example 17

```
ALTER TABLE s_employees ADD VERSIONING USE HISTORY TABLE v_employees;
```

The CREATE TABLE statement in Example 16 creates the new table called **v_employees**, which has the same structure as the **s_employees** table and is used to store old versions of rows. Example 17 modifies the structure of the base table to enable it for versioning and to identify the versioning table.

B. Retrieving and Modifying Data from Tables with System Time

The INSERT statement for tables with system time has the same syntax and semantics as the corresponding statement for system-versioning tables. IBM DB2 supports the same syntax for the UPDATE and DELETE statements as SQL:2011, but the semantics of these operations is different: The modification of a row using the UPDATE statement is maintained so that the new version of the row is placed in the base table and the corresponding old version in the versioning table. Similarly, during deletion, the data from the base table is deleted and copied in the corresponding versioning table. The system sets the end time of the deleted data in the versioning table to the transaction start time of the DELETE statement.

6 EVALUATION AND CONCLUSIONS

All temporal data models can be evaluated in relation to several concepts. In this article, we will evaluate the temporal data model introduced in the SQL:2011 specification in relation to the three most important concepts:

- Time dimensions
- Implicit vs. explicit timestamps
- Grouping of time-varying attributes

6.1 Time Dimensions

The most important concept of temporal data models is time dimension, and there are three different forms of it: valid time, transaction time and bitemporal. Valid time concerns the time when a fact is true in the real world. For this reason, this form of time is independent of its storage in a database and can concern the past, present and future snapshots of the fact.

Transaction time concerns the time the event was present in the database as stored data. The transaction time of an event describes the times, where the event is stored in a database and presents the correct image of the modelled world. Timestamps of transaction time events are defined according to the schedule adopted by the operating system. Therefore, we can build the history of all such timestamps in relation to the past and current time, but not in relation to future. Additionally, only the current values may be updated, and the updates cannot be retroactive.

The union of both forms explained above is called bitemporal.(There is also a special case of bitemporal model, when the valid and transaction times of a fact are identical. As a simple example for this case, the situation where a fact is recorded as soon as it becomes valid in reality can be considered.)

The most temporal data models proposed in the literature support only valid time. The specification of temporal data in SQL:2011 supports all three dimensions.

6.2 Implicit vs. explicit timestamps

The difference between implicit and explicit timestamps concerns how the association of times is represented. In case of explicit timestamps this association is represented by fully explicit timestamp attributes. This issue has consequences in relation to update languages in the following way: While transaction times of facts are supplied by the system itself, update operations in transaction-time models treat the temporal aspect of facts implicitly. On the other hand, the user is responsible to supply valid times of facts. Therefore, updating facts in valid time and bitemporal data models generally must treat time explicitly and are forced to represent a choice as to how the valid times of facts should be specified by the user. The SQL:2011 specification supports explicit and implicit timestamps in the way described above: In the case of valid time (in application-time period tables), the user is in charge of supplying values for begin and end of the particular time period. The values of transaction time periods in system-versioned tables are automatically set by the system.

6.3 Grouping of Time-Variant Attributes

Temporal data models support two different approaches in relation how time-variant attributes are attached: tuple time stamping and attribute time stamping. In tuple time stamping, each tuple is augmented by one or two attributes for the recording of timestamps. One additional attribute can be used to record either the time point at which the tuple becomes valid or the time at which the data is valid. Two additional attributes are used to record the start and end time points of the corresponding time interval of validity of the corresponding data. Tuple time stamping is usually applied in temporal relational data models, meaning that the first normal form (1NF) has to be maintained.

The second approach, attribute time stamping, means that the time is associated with every attribute which is time-varying. Therefore, a history is formed for each time-varying attribute within each tuple. As a result, the degree of the relation is reduced by one or two compared with the tuple time stamping, since timestamps are part of the attribute values. Values in a tuple which are not affected by a modification do not have to be repeated. So, the history of values is stored separately for each attribute.

Each of the two approaches has benefits and disadvantages. Tuple time stamping, which implies 1NF may introduce redundancy because attribute values that change at different times are repeated in multiple tuples. On the other hand, temporal relational models can use only this approach. The attribute time stamping overcomes the disadvantage of data redundancy introduced when applying tuple time stamping, but it cannot directly use existing relational storage structures or query evaluation techniques that depend on atomic attribute values. The specification of temporal data in SQL:2011 uses tuple time stamping, because SQL is the language for relational database systems.

6.4 Deficiencies of the Specification

The main deficiency of the standardized specification for temporal data is the omission to support the PERIOD data type. A period can be defined as a duration that represents a set of contiguous time units within the duration. It has a beginning and ending bound. The both are defined by the value of two elements: a beginning element and an ending element. Beginning and ending elements can be DATE, TIME, or TIMESTAMP types, but both must be of the same type. The main advantage of the PERIOD data type is that it is naturally (and very easy) to define operations on such a data type. For instance, the following operations are concerned as operations on time periods: CONTAINS, EQUALS, PRECEDES, SUCCEEDS and OVERLAPS.

There are several other deficiencies, which are listed below:

- Coalescing is not supported
- Temporal joins are not supported
- Multiple application-time periods per table are not supported

Coalescing is similar operation to the elimination of duplicates in conventional databases. The aim of coalescing is to merge bring together tuples with identical attribute values and with timestamps, which are adjacent in tine, or share some time periods in common. Temporal join means that a row from one table is joined with a row from another table such that their application-time or system-time periods satisfy a condition. (The notion of multiple application-time periods is obvious per se.)

REFERENCES

[1] Darwen, H.; Date, C.J. (2006) - An overview and Analysis of Proposals Based on the TSQL2 Approach, in Date on Database: Writings 2000-2006, C.J. Date, Apress, 2006.

[2] Kulkarni, K (2012) - Temporal Features in SQL Standard, in http://metadata-standards.org/Document-library/Documents-by-number

[3] Lorentzos, N. (1993) - The Interval-extended Relational Model and Its Applications to Valid-time Databases, in Temporal Databases.

[4] Melton,J. (2001) - SQL:1999, Understanding Relational Language Components, Morgan-Kaufman

[5] Saracco, C.M.; Nicola, M.; Gandhi, L.(2012) - A matter of time: Temporal data management in DB2, in www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf

[6] Snodgrass, R.T. et al (1994) - TSQL2 Language Specification, in *SIGMOD Record* 23(1).

[7] Snodgrass, R.T (1995) - The TSQL2 Temporal Query Language, Springer Verlag.

[8] SQL:2001 Standard (2011) - ISO/IEC 9075-2:2011, Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation).

[9] Zemke, F. (2012) - What's New in SQL:2011, SIGMOD Record.