Support of Temporal Data in Database Systems

Dušan Petković
University of Applied Sciences
Hochschulstr. 1,
Rosenheim, 83024, Germany

ABSTRACT
The time is generally a challenging task. All issues in relation to time can be better supported using temporal data models. Almost all enterprise database systems have implemented temporal data, partly according to the model specified in the SQL:2011 standard and partly according to other, older temporal models. In this article five temporal concepts will be used to investigate their implementations in enterprise database systems. Also, strengths and weaknesses of these implementations will be discussed.

Keywords
Database systems, temporal model, PERIOD type

1 INTRODUCTION
Research of temporal data in relation to database systems has a very long history. For this reason, many temporal data models have been introduced. The current versions of enterprise database systems support either the temporal model of the SQL:2011 standard [4] or have implemented concepts according to other temporal models. Hence, evaluation of the following general concepts will be done:

- Time dimensions
- Temporal key constraints
- Coalescing
- PERIOD data type
- Implicit vs. explicit timestamps

Note that temporal joins are not discussed in the paper, because none of systems, mentioned in this article supports the feature.

1.1 Time Dimensions
There are three different forms of time dimensions: user-defined time, valid time, and transaction time. User-defined time is a time representation designed to meet specific needs of users. Valid time concerns the time when an event is true in the real world. For this reason, an event is independent of time when it is stored and can concern past, present and future snapshots of it.

Transaction time concerns the time when an event was present in the database as stored data. Therefore, transaction time of an event presents the correct database image of the modelled world. Timestamps of transaction time are defined according to the schedule adopted by the operating system. According to this, the history of all such timestamps in relation to the past and current time can be built. (Only current values may be updated, and the updates cannot be retroactive, as in case of valid time.)

A data model that supports only valid time is called valid-time model and one that supports only transaction time is called transaction-time model. If a data model supports both of them, it is called bitemporal.

1.2 Temporal Key Constraints
There are two temporal key constraints: one in relation to primary key (PK) and the other in relation to referential integrity (RI). The requirements concerning temporal primary key depend whether the table captures valid or transaction time. In the case of valid time, the convenient primary key of relational tables is not sufficient and the primary key has to include time-variant attributes, because each value of the relational PK must be unique at any given point in valid time. On the other hand, tables capturing transaction time always include only one tuple concerning current time, and (possibly) several history rows, which cannot be modified. For this reason, in case of transaction time, relational primary key is also the temporal primary key of the corresponding table with time-variant attributes.

In the case of temporal referential integrity, each value of the relational foreign key in the child table must correspond to some value of the relational primary key in the parent table at any given point in time. Therefore, it must be possible to forbid a tuple in a child table whose valid time is not contained in the time period of the corresponding tuple in the parent table [14].

1.3 Value Equivalence and Coalescing
Two tuples are value equivalent if they are identical. A tuple is coalesced, if overlapped or consecutive value-equivalent tuples are disallowed. When timestamps of tuples have temporal elements as values, the requirement of coalescing is identical to the requirement that there will be no value-equivalent tuples [3].

The need for coalescing happens when a projection or union operation is performed during retrieval of data or INSERT i.e. UPDATE statements are executed. The general approach for implementation of coalescing is similar to the problem of computation of the transitive closure of a graph, with the subsequent deletion of non-maximal paths and can be implemented by iterating an INSERT statement that coalesces two paths and inserts a new tuple into the relation [15].

1.4 PERIOD Data Type
The PERIOD data type is specified as a time interval which comprises the set of subsequent time units. These units use a closed-open concept, meaning that the starting granule of the time period is included, while the end granule is excluded. The data type of the start and end of the period can be DATE, TIME or TIMESTAMP. The main advantage of the PERIOD data type is that it can be used in natural way to represent time intervals, the same way as, for instance, the GEOMETRY data type is used to represent spatial data.

The use of the PERIOD data type requires specification and implementation of corresponding functions i.e. methods. A temporal constructor with the same name as the data type is implicitly defined, when a time-variant column of that type is specified. (The phrase “time-variant column” will be used for columns, which contain time-variant data.) A database sys-

tem supporting this data type usually implements several temporal functions, such as CONTAINS and OVERLAPS. A proposal, how the PERIOD type can be specified, is given in [13]. Functions of this data type are based upon Allen's operators [2].

1.5 Implicit vs. Explicit Timestamps

The association of time with temporal facts is different for existing temporal data models. Some of them specify that this association is explicit, meaning that temporal facts are handled in the same way as all other table's columns. Other models treat time-variant columns as special columns, which are not handled in the same way as other, temporal-invariant attributes. This issue has also 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 must treat time explicitly and are forced to represent a choice how valid times of facts should be specified by the user.

1.6 Roadmap

The rest of the paper is organized as follows. Section 2 describes the implementation of temporal concepts in the IBM DB2 database system, while Section 3 explains how these concepts are implemented in Teradata. The next two sections discuss concepts supported in Oracle and MS SQL Server, respectively. The structure of these sections is identical to the structure of the previous two. The last section summarizes the results and shows our conclusions.

PostgresSQL has not been considered in this work, because it does not support temporal concepts discussed in this article.

2 IBM DB2

IBM DB2 supports temporal data since Version 10. The syntax and semantics of the underlying model is, besides a couple of insignificant differences, identical to the temporal model specified in the SQL:2011 standard. One of these differences is that valid time is called „business time" and „system time" is a DB2 phrase for transaction time. The SQL:2011 standard calls tables with valid time "application-time period tables" and "system-versioned tables" are tables, which contain transaction time [7].

2.1 IBM DB2: Time Dimensions

DB2 supports valid time as well as transaction time. The union of valid and transaction time is supported, too. Besides a few small differences, the IBM implementation is based upon the standard [9].

2.1.1 Support of Valid Time

Tables with valid time contain two time-variant columns, one for the start and the other for the end of valid time. These columns can be either of DATE or TIMESTAMP type. The CREATE (ALTER) TABLE statement is extended with the PERIOD clause, which specifies the time interval.

Example 1

```
CREATE TABLE V_Emp(
ENo INT NOT NULL, EDept INT,
EStart DATE NOT NULL, EEnd DATE NOT NULL,
    PERIOD BUSINESS_TIME (EStart, EEnd),
    PRIMARY KEY(ENo,BUSINESS_TIME
                WITHOUT OVERLAPS));
```

Time-variant columns EStart and EEnd are explicitly specified in the PERIOD clause. Additionally, the clause defines implicitly that startdate < enddate. In contrast to the SQL:2011 specification, the name for the PERIOD clause is given and cannot be specified by the user.

The insertion of tuples in a table with valid time corresponds to the convenient insertion of rows in a relational table. The following example shows such INSERT statements and the content of the V_Emp table after insertion.

Example 2

```
INSERT INTO V_Emp(Eno, Edept, EStart,
Eend)
VALUES(12345,3,'01.01.2013',
'01.01.2014');
INSERT INTO V_Emp(Eno, Edept, EStart,
Eend)
VALUES(12345,4,'01.01.2014',
'31.12.9999');
```

ENO	EDEPT	ESTART	EEND
12345	3	2013-01-01	2014-01-01
12345	4	2014-01-01	9999-12-31

The value '31.12.9999' in the second INSERT statement specifies the explicit value for the end of time interval, which is called „forever". In contrast to other database systems, which support temporal data, IBM DB2 does not have any reserved keyword for "forever".

Besides the convenient syntax for the UPDATE statement, IBM DB2 supports the FOR PORTION clause, which specifies an additional temporal condition. In other words, only such tuples are updated, where the considered time interval contains or overlaps with the one specified in the clause.

Example 3

```
UPDATE V_Emp FOR PORTION OF BUSINESS_TIME
    FROM '01.07.2013' TO '01.07.2014'
    SET EDept = 4   WHERE Eno = 12345;
```

ENO	EDEPT	ESTART	EEND
12345	4	2013-07-01	2014-01-01
12345	4	2014-01-01	2014-07-01
12345	3	2013-01-01	2013-07-01
12345	4	2014-07-01	9999-12-31

The time period specified in the FOR PORTION clause overlaps the time periods of both tuples from the result of Example 2. For this reason, these tuples will be modified in the following way: the column with the value Edept=4 will be divided in two columns, one with the time period (1.1.2014, 1.7.2014) and the other with the time period (1.7.2014, 31.12.9999). (The column with the value **Edept**=3 is modified in the similar way.) The FOR PORTION clause can be also used with DELETE.

Example 4

```
DELETE FROM V_Emp
 FOR PORTION OF BUSINESS_TIME
    FROM '01.06.2013' TO '01.02.2014'
WHERE Eno = 12345;
ENO   EDEPT ESTART     EEND
 ----- ----- ---------- ----------
 12345     3 2013-01-01 2013-06-01
 12345     4 2014-02-01 2014-07-01
 12345     4 2014-07-01 9999-12-31
```

The DELETE statement above deletes the first tuple from the result of Example 3, because the time period of the tuple is contained in the specified time period of the FOR PORTION clause. The second and the third tuple in that result have the time periods which overlap the specified one. The system will execute DELETE so, that the overlapped period of the tuples will be deleted and the rest of information will be saved.

IBM DB2 uses the convenient syntax to query a table with valid time. Additionally, there are three options that are part of the FROM clause:

a) FOR BUSINESS_TIME AS OF …

b) FOR BUSINESS_TIME BETWEEN … AND …

c) FOR BUSINESS_TIME FROM … TO …

The first form of this clause displays all tuples with a time interval that contains the specified time granule. In contrast to the first form, the second and the third form specify time periods. The BETWEEN … AND … form specifies a closed time interval, while the FROM … TO … form defines a closed-open one. In both cases, all tuples with time periods, which overlaps the specified one are selected. (Example 9 shows queries that use clauses with the same semantics, but for transaction time.)

2.1.2 Support of Transaction Time

IBM DB2 supports transaction time with so called tables with system time. Similarly to valid time, these tables include two columns, one for the start and the other for the end of transaction time. (The data type of these columns has to be TIMESTAMP.)

Example 5

```
CREATE TABLE T_Emp
(ENo INT PRIMARY KEY NOT NULL,  EDept INT,
  sys_start    TIMESTAMP(12)    GENERATED ALWAYS
          AS ROW BEGIN NOT NULL,
  sys_end TIMESTAMP(12) GENERATED ALWAYS
          AS ROW END NOT NULL,
  t_start TIMESTAMP(12) GENERATED ALWAYS
AS TRANSACTION START ID IMPLICITLY HIDDEN,
PERIOD SYSTEM_TIME (sys_start, sys_end));
```

The specification of time-variant attributes contains additional clauses. The GENERATED ALWAYS AS ROW BEGIN and GENERATED ALWAYS AS ROW END clauses specify the columns for the start and end time, respectively. The GENERATED ALWAYS AS TRANSACTION START clause specifies the transaction start time. The IMPLICITLY HIDDEN clause will be discussed later.

The definition of transaction time involves altogether two tables: The first one stores only tuples with current time, while the second one, called history table, stores old versions of current tuples. Therefore, there are three steps in creation of tables with transaction time:

a) Create a base table

b) Create a history table

c) Alter the base table to enable versioning

Example 6

```
CREATE TABLE T_Emp_history LIKE T_Emp;
ALTER TABLE T_Emp ADD VERSIONING
    USE HISTORY TABLE T_Emp_history;
```

The history table, **T_Emp_history,** has the same structure as the base table. The second statement in Example 6 acitivates versioning for the base table and identifies the history table as the one, where old versions of tuples will be stored.

While the values for time-variant columns will be implicitly stored by the system, the INSERT statement in relation to tables with transaction time provides values of time-invariant columns. Example 7 shows two INSERT statements and the content of the **T_Emp** table after insertion. These INSERT statements are executed on 22.4.2014. For simplicity, all results in the following text concerning transaction time display only the date portion of the time-variant columns.

Example 7

```
INSERT INTO T_Emp(Eno, Edept)
  VALUES(12345,3);
INSERT INTO T_Emp (Eno, Edept)
  VALUES (1235, 4);
```

ENO	EDEPT	SYS_START	SYS_END
12345	3	2014-04-22	9999-12-31
1235	4	2014-04-22	9999-12-31

The UPDATE statement in the example is executed two days later.

Example 8

```
UPDATE T_Emp SET Edept = 4
      WHERE ENo = 12345;
```

(The content of the base table)

ENO	EDEPT	SYS_START	SYS_END
12345	4	2014-04-24	9999-12-31
1235	4	2014-04-22	9999-12-31

(The content of the history table)

ENO	EDEPT	SYS_START	SYS_END
12345	3	2014-04-22	2014-04-24

In case of transaction time only current tuples can be deleted. Therefore, a DELETE statement corresponds to the UPDATE statement and the „deleted" tuples are moved from the current to the corresponding history table.

The syntax and semantics of convenient SELECT statements remain unchanged in relation to transaction time. To query old versions of tuples, DB2 supports three options in the FROM clause of the SELECT statement:

a) FOR SYSTEM_TIME AS OF …

b) FOR SYSTEM_TIME BETWEEN … AND …

c) FOR SYSTEM_TIME FROM … TO …

The semantics of these options is equivalent to the semantics of the corresponding FOR PORTION options, which are described in Section 2.1.1.

Example 9

```
SELECT Edept FROM T_Emp
FOR SYSTEM_TIME AS OF DATE '2012-08-07'
        WHERE Eno = 12345;
SELECT abt_id  FROM T_Emp
FOR SYSTEM_TIME
FROM DATE'2014-04-22' TO DATE'2014-04-24'
    WHERE Eno = 12345;
```

2.1.3 Bitemporal Tables

A bitemporal table is a union of tables with valid and transaction time.

Example 10

```
CREATE TABLE BI_Emp(
  ENo INT NOT NULL, EDept INT,
  EStart DATE NOT NULL, EEnd DATE NOT
NULL,
  sys_start    TIMESTAMP(12)    GENERATED
ALWAYS
               AS ROW BEGIN NOT NULL,
  sys_end TIMESTAMP(12) GENERATED ALWAYS
               AS ROW END NOT NULL,
   t_start TIMESTAMP(12) GENERATED ALWAYS
AS  TRANSACTION  START  ID  IMPLICITLY
HIDDEN,
  PERIOD      SYSTEM_TIME      (sys_start,
sys_end),
    PERIOD BUSINESS_TIME (EStart, EEnd),
    PRIMARY  KEY  (ENo,  BUSINESS_TIME
WITHOUT OVERLAPS));
```

After creation of a bitemporal table, the corresponding history table must be created and the current table has to be enabled for versioning. After that, all DML statements concerning valid and transaction time can be used.

2.2 IBM DB2: Temporal Key Constraints

IBM DB2 supports the WITHOUT OVERLAPS clause in the PRIMARY KEY option of the CREATE (ALTER) TABLE statement. This clause ensures that each value of the relational primary key is unique at any given point in valid time (see Example 1).

Example 11

```
INSERT INTO V_Emp(Eno,Edept,EStart,Eend)

VALUES(12345,5,'01.03.2013','01.06.2013')
;
```

The execution of the INSERT statement in the example above will be rejected by the system, because the specified time period ('01.03.2013', '01.06.2013') of the employee with ID=12345 overlaps the time period of an existing tuple. (The execution of similar UPDATE statements will be rejected by the system, too.) IBM DB2 does not support referential constraints on tables with valid time. As described in [5], these constraints can be implemented using triggers or stored procedures.

2.3 Coalescing and PERIOD Data Type

IBM DB2 does not support coalescing. This can be seen from the result of Example 3: The result should contain only two tuples, because the first, second and the fourth tuple are consecutive tuples and logically represent a single tuple.

IBM DB2 does not support the PERIOD data type. Hence, users have to specify additional time-variant columns for valid as well as transaction time.

2.4 Implicit vs. explicit timestamps

IBM DB2 supports explicit as well as implicit timestamps. Explicit timestamps are supported by default, while implicit timestamps can be specified using the IMPLICITLY HIDDEN option (see Example 5). In that case, "SELECT * FROM table_name" does not display values of any time-variant column. The only way how these values can be displayed is naming them in the corresponding SELECT list of a query.

3 TERADATA

Teradata supports temporal data since Version 10. Versions 10.1 and 11 contain several new improvements in relation to this issue. Teradata's temporal model is based upon the TSQL2 model [11] and contains three general features: time dimensions, the PERIOD data type and temporal qualifiers.

3.1 Teradata: Time Dimensions

3.1.1 Support of Valid Time

Time-variant attributes in Teradata's valid time tables are specified using the PERIOD data type. The definition of an attribute of the PERIOD type also requires the additional specification, which can be DATE or TIMESTAMP.

Example 12

```
CREATE MULTISET TABLE Emp(
    ENo INT NOT NULL, EDept INT NOT NULL,
```

```
    Emp_period PERIOD(DATE)AS VALIDTIME)
         PRIMARY INDEX (ENo);
CREATE MULTISET TABLE Dept(
    DNo INT NOT NULL, DName VARCHAR(30),
    Dept_period      PERIOD(DATE)      AS
VALIDTIME)
         PRIMARY INDEX (DNo);
```

The **Emp** table contains the time-variant attribute **Emp_Period**, of the PERIOD data type. The VALIDTIME option specifies the attribute as valid time period.

All Teradata's DML statements can contain temporal qualifiers, which are used for specification of conditions in relation to time-variant columns. The following qualifiers exist:

a) CURRENT

b) AS OF

c) SEQUENCED

d) NONSEQUENCED

All qualifiers above have two forms: one with the VALIDTIME keyword (for valid time), and one with the TRANSACTION keyword (for transaction time). The CURRENT qualifier selects only the current tuples, i.e. tuples with valid time values related to the current time. (CURRENT VALIDTIME is the default temporal qualifier for valid time.) The SEQUENCED qualifier selects the tuples with the time interval that is contained in the time period specified with the qualifier. The NONSEQUENCED qualifier specifies that the time dimension will be ignored and the involved table is considered as a non-temporal table. A query with AS OF expression retrieves tuples where the time period overlaps the time period of the specified expression.

Note, that the AS OF expression is a generalization of the CURRENT qualifier, when the time granule is not the current time. Therefore, CURRENT corresponds to AS OF CURRENT_DATE i.e. AS OF CURRENT_TIMESTAMP.

Example 13

```
SEQUENCED VALIDTIME
INSERT INTO Emp (Eno, EDept, Emp_period)
VALUES(12345,3,PERIOD(DATE'2013-01-01',
                      DATE'2014-01-01'));
SEQUENCED VALIDTIME
INSERT INTO Emp (Eno, EDept, Emp_period)
VALUES (12345, 4,PERIOD(DATE '2014-01-
01', UNTIL_CHANGED));
```

The INSERT statement in example above contains values of the DATE type, which define the start and end of valid time. For this reason each INSERT statement in Example 13 must be prefixed with the SEQUENCED VALIDTIME clause. Teradata supports the UNTIL_CHANGED keyword for „forever", as can be seen from the second statement of the example.

The UPDATE and DELETE statements in Examples 14 and 15 are semantically equivalent to UPDATE and DELETE in Examples 3 and 4, respectively.

Example 14

```
SEQUENCED VALIDTIME
PERIOD(DATE'2013-07-01',DATE'2014-07-01')
  UPDATE Emp SET Edept=4 WHERE Eno=12345;
```

Example 15

```
SEQUENCED VALIDTIME
PERIOD(DATE  '2013-06-01',DATE  '2014-02-
01')
DELETE FROM Emp WHERE Eno = 12345;
```

All valid time queries can contain any of the four already mentioned temporal qualifiers. Example 16 shows the use of the SEQUENCED qualifier.

Example 16

```
SEQUENCED VALIDTIME
PERIOD(DATE  '2013-03-01',DATE  '2013-05-
01')
SELECT * FROM Emp  WHERE Eno = 12345;
```

The Teradata's AS OF qualifier in relation to queries corresponds logically to the DB2 option with the same name. Also, the SEQUENCED qualifier corresponds to the BUSINESS_TIME FROM … TO … option in DB2. (The NONSEQUENCED qualifier does not have any logical equivalent in IBM DB2.)

3.1.2 Support of Transaction Time

The TRANSACTIONTIME clause defines a time period as transaction time. The transaction time-variant column must be of theTIMESTAMP WITH TIME ZONE data type.

Example 17

```
  CREATE MULTISET TABLE T_Emp
(ENo INT NOT NULL,EDept INT NOT NULL,
  T_Emp_period  PERIOD(TIMESTAMP(6)  WITH
TIME ZONE) NOT NULL AS TRANSACTIONTIME)
    PRIMARY INDEX (ENo);
```

As in case of valid time, Teradata supports temporal qualifiers for transaction time. All four qualifiers described in the section concerning valid time can be used for transaction time. Their meaning is similar to the meaning of the corresponding valid time qualifiers.

The syntax and semantics of INSERT, UPDATE and DELETE statements for transaction time are simpler than for valid time, because these statements can be applied only to current tuples. For this reason the syntax of INSERT and UPDATE statements in Teradata is identical to the syntax of the same statements in IBM DB2.

All Teradata's queries in relation to transaction time are based upon the convenient syntax. Teradata supports temporal extensions for SELECT, semantically similar to those extensions defined in IBM DB2. These extensions allow queries against current tuples as well as old versions of them. Teradata supports two temporal options:

a) CURRENT TRANSACTIONTIME

b) TRANSACTIONTIME AS OF TIMESTAMP

The first option selects only the current tuples, which fulfill the specified condition. Therefore, this option corresponds semantically to the FOR SYSTEM_TIME AS OF CURRENT_DATE clause in DB2. The semantics of the second option above is identical to the semantics of the DB2's FOR SYSTEM_TIME AS OF clause. The query in Example 18 displays all tuples which contain the specified timestamp.

Example 18

```
TRANSACTIONTIME AS OF
  TIMESTAMP '2010-01-01 23:59:59'
SELECT Eno, EDept FROM T_Emp;
```

Note that Teradata does not support the SEQUENCED TRANSACTIONTIME clause. This clause can be implemented using the NONSEQUENCED TRANSACTIONTIME clause and the OVERLAPS operator [12].

3.1.3 Bitemporal Tables

Teradata supports bitemporal tables, too.

Example 19

```
CREATE  MULTISET TABLE Bi_Emp(
ENo INT NOT NULL, EDept INT NOT NULL,
Emp_VT    PERIOD(DATE)    NOT    NULL    AS
VALIDTIME,
Emp_TT  PERIOD(DATE  WITH  TIME  ZONE)NOT
NULL
  AS   TRANSACTIONTIME)   PRIMARY   INDEX
(ENo);
```

3.2 Teradata: Temporal Key Constraints

Teradata supports three qualifiers in relation to temporal primary key constraint:

a) CURRENT VALIDTIME PRIMARY KEY

b) SEQUENCED VALIDTIME PRIMARY KEY

c) NONSEQUENCED VALIDTIME PRIMARY KEY

The CURRENT VALIDTIME PRIMARY KEY constraint ensures that the value for the constrained column in a tuple is unique for all instances of time from current time through the future. Current and future tuples that have overlapping time periods are prevented from having the same value in the constrained columns. The SEQUENCED VALIDTIME PRIMARY KEY constraint ensures that the value for the constrained column in a tuple is unique for all instances of time, including past, current, and future. Any tuples that have overlapping time periods are prevented from having the same value in the constrained columns. The semantics of this constraint is identical to the semantics of the DB2's WITH OVERLAPS clause. The NONSEQUENCED VALIDTIME PRIMARY KEY constraint treats a time-variant column as a convenient column. This constraint ensures that the value for the constrained column in a tuple is unique amongst all tuples in the table.

Teradata does not support referential constraints on tables with valid time. A discussion concerning Teradata's temporal referential integrity can be found in [10].

3.3 Teradata: PERIOD Data Type

Teradata is the only one DBMS which, at this moment, supports the PERIOD type (see Example 12). Additionally, Teradata supports several temporal operators, such as: CONTAINS, OVERLAPS, PRECEDES, SUCCEDES and MEETS [13].

3.4 Teradata: Coalescing

Teradata does not support coalescing. The only way to coalesce data can be done using the temporal function called P_NORMALIZE. This function does not concern the storage of data, it just displays data in coalesced form.

Note that coalescing in Teradata will be implemented in one of future versions, using analytical functions. The description of this feature can be found in [1].

3.5 Implicit vs. Explicit Timestamps

Teradata supports implicit timestamps by default. This means that values of temporal attributes will not be displayed with queries, such as the following one: "SELECT * FROM table_name". The only way how values of time-variant columns can be displayed is naming them explicitly in the SELECT list:

```
SELECT Emp.* , Emp_period FROM Emp;
```

4 ORACLE

Oracle supports temporal data since Version 12c. The characteristic of Oracle's support for temporal data is that there are two independent components, where each of them supports a single time dimension. Valid time is based upon the component called "Temporal Validity", while the "Flashback Data Archive" component covers transaction time.

4.1 Oracle: Time Dimensions

4.1.1 Support of Valid Time

The PERIOD clause in the CREATE TABLE statement is used to specify valid time intervals [6]. Whether time-variant attributes are implicitly or explicitly defined depends on how the PERIOD clause is specified.

Example 20

```
CREATE TABLE Emp(ENo INT, EStart DATE,
    EEnd DATE, EDept INT,
PERIOD FOR EPeriod (EStart, EEnd));
CREATE TABLE Emp_1(ENo INT,  EDept INT);
ALTER  TABLE  Emp_1  ADD  PERIOD  FOR  valid_time
```

The names of the two time-variant attributes, **Estart** and **EEnd,** are explicitly defined in the PERIOD clause. The second statement creates the **Emp_1** table with implicitly defined time-variant attributes. The names of these attributes, **valid_time_start** and **valid_time_end,** are derived by the system from the name of the PERIOD clause.

Example 21

```
INSERT  INTO  Emp  VALUES  (12345,  TO_DATE
('01-JAN-2013'),TO_DATE('01-JAN-
2014'),3);
INSERT INTO Emp
VALUES(12345,TO_DATE('01-JAN-
2014'),NULL,4)
INSERT              INTO              Emp_1
(Eno,valid_time_start,valid_time_end,EDept)    VALUES(12345,TO_DATE('01-JAN-2013'),
TO_DATE('01-JAN-2014'),3);
```

In contrast to the first two statements in the example above, the last one inserts a tuple in the **Emp_1** table. In the latter case, the list of all names of implicitly defined tables's columns must be provided.

Note that the second INSERT statement uses the NULL value, which has a special meaning in the Oracle's temporal model: The specification of NULL for end of time period of valid time means "forever". (NULL could be also used for the start of time period of valid time, meaning "since begin of time measurement".)

In contrast to IBM DB2 and Teradata, Oracle does not support temporal extensions for the UPDATE statement to specify a valid time interval. The same is true for DELETE.

4.1.2 Support of Transaction Time

The creation of such a table requires that an archive is created first, using CREATE FLASHBACK ARCHIVE statement.

Example 22

```
CREATE TABLE T_Emp
(Eno INT PRIMARY KEY NOT NULL,  Edept INT)
     FLASHBACK ARCHIVE dusan;
```

The insertion of tuples in a table with transaction time is identical to the same activity with IBM DB2 (see Example 7). The same is true for the UPDATE statement (see Example 8).

Queries upon tables with transaction time use the convenient SELECT statement. Oracle supports additionally two different forms of the TIMESTAMP option to select current as well as old versions of tuples. To specify a time granule, the AS OF TIMESTAMP clause is used (see the first statement in the following example). The second query in the same example uses VERSIONS BETWEEN TIMESTAMP. Therefore, both queries in Example 26 correspond semantically to statements in Example 9, respectively. (Columns **versions_startscn** and **versions_endscn** are pseudo columns that are used to log information in relation to the creation i.e. "deletion" of a tuple.)

Example 23

```
SELECT * FROM T_Emp
    AS OF TIMESTAMP CURRENT_TIMESTAMP;
SELECT * FROM T_Emp AS OF TIMESTAMP
  TO_TIMESTAMP('12-05-2014 11:20',
              'dd-mm-yyyy hh24:mi');
SELECT eno , versions_startscn,
 versions_endscn  FROM T_emp
          VERSIONS BETWEEN TIMESTAMP
  TO_TIMESTAMP('12-05-2014         11:20',
 'dd-mm-yyyy hh24:mi') AND
 TO_TIMESTAMP('12-05-2014         11:30',
 'dd-mm-yyyy hh24:mi');
```

4.1.3 Bitemporal Tables

A bitemporal table in Oracle is formed through union of a valid time and a transaction time table.

4.2 Temporal Key Constraints, Coalescing and the PERIOD Data Type

Oracle support neither temporal key constraints nor coalescing. As can be seen from Example 20, users have to define time-variant columns for valid as well as transaction time, because the PERIOD data type is not supported.

4.3 Implicit vs. Explicit Timestamps

As can be seen from Example 23, Oracle supports implicit as well as explicit timestamps. To specify explicit timestamps, the time-variant columns have to be defined and the PERIOD clause must contain their names. If the PERIOD clause does not contain the specification of time-variant columns their names are derived from the name of this clause.

Example 24

```
SELECT * FROM Emp_1;
SELECT
e.*,valid_time_start,valid_time_end
   FROM Emp_1 e;
```

Queries in the example above display different results. The first one displays values of time-invariant columns, while the second one displays values of all columns.

5 MS SQL SERVER

Microsoft supports temporal data in SQL Server 2016. The characteristic of Microsoft's support is that there are just a few concepts which the system supports. For this reason, the structure of this chapter will be different than the structure of the others. The detailed discussion of the future support of temporal data for SQL Server can be found in [8]. From all concepts listed in the introductory part of the paper, SQL Server supports transaction time and implicit i.e. explicit timestamping, only.

Example 25

```
CREATE TABLE dept_temp
(dept_no  CHAR(4)  NOT  NULL  PRIMARY  KEY
CLUSTERED,
dept_name  CHAR(25)  NOT  NULL,  location
CHAR(30), start_date DATETIME2 GENERATED
ALWAYS AS ROW START HIDDEN NOT NULL,
  end_date  DATETIME2  GENERATED
ALWAYS AS ROW END HIDDEN NOT NULL,
PERIOD FOR SYSTEM_TIME(start_d, end_d))
       WITH(SYSTEM_VERSIONING =
    ON(HISTORY_TABLE=dbo.Dept_History));
```

Microsoft's implementation of transaction time corresponds to specification of the SQL:2011 standard. The WITH SYSTEM_VERSIONING option is used to enable the creation of the corresponding history table. Microsoft refers to the tables that implement transaction time as "temporal tables".

SQL Server supports implicit as well as explicit timestamps. To specify timestamps as implicit ones, the HIDDEN option must be used (see example above). The queries in example 24 (for Oracle) have the same semantics for SQL Server, too.

The only temporal extensions are in relation to the SELECT statement. These are:

- AS OF <date_time>
- FROM <start_time> TO <end_time>
- BETWEEN <start_time> AND<end_time>
- CONTAINED IN (<start_time>,<end_time>)
- ALL

The first three clauses correspond to the clauses specified in the SQL standard, while the last bust one returns a result with the values of old versions that were opened and closed within the time range defined by the two parameters in CONTAINED. ALL queries current and historical data without any restrictions.

6 SUMMARY

The most important attitude of temporal extensions in DB2 is that they are implemented according to the specification in the SQL:2011 standard. The only significant difference is that IBM DB2 uses two tables to store transaction time data, one for current tuples and the other for old versions of them. In contrast to IBM DB2, the temporal model of Teradata corresponds to the TSQL2 model. Concerning implicit and explicit timestamping, Teradata supports only the first one. As a direct consequence of this fact, the basic data object in Teradata is not a relation, and the temporal model supported by this database system is certainly not relational. Oracle's current implementation of temporal data seems rudimentary, because many temporal concepts are not implemented.

Two most important temporal concepts are the support of the PERIOD data type and coalescing. Teradata has already implemented the first concept and will implement the second one in the future. IBM DB2, and its underlying temporal model specified in the SQL:2011 standard lack these concepts.

7 REFERENCES

[1] Al-Kateb, M.; Ghazal, A.; CROLOTTE, A. 2012. An Efficient SQL Rewrite Approach for Temporal Coalescing in Teradata, Database and Expert Systems Appl., LNCS **7447**.

[2] Allen, J. F. 1983. Maintaining knowledge about temporal intervals, CACM, 832–843

[3] Böhlen, M.; Snodgrass, R.T. and Soo, M. 1996. Coalescing in Temporal Databases, Proc. of 22th Int. Conf. on VLDB

[4] ISO/IEC 9075-2:2011, 2011.Information technology–DB languages–SQL:Part 2.

[5] Nicola, M.; Sommerlandt, M., 2012. Managing time in DB2 with temporal consistency,http://www.ibm.com/developerworks/data/library/techarticle/dm-1207db2temporalintegrity/index.html

[6] Oracle, 2014. Multi-temporal Features in Oracle 12c, http://www.salvis.com/blog/2014/01/04/multi-temporal-database-features-in-oracle-12c/

[7] Petković, D., 2013. Was lange währt, wird endlich gut: Temporale Daten im SQL-Standard,DB-Spektrum,**13, 131-8**.

[8] Petković, D. 2016. SQL Server 2016: A Beginner's Guide, Mc-Graw Hill.

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

[10] Sannik, G. and Daniels, F. 2012. Enabling the Temporal Data Warehouse, http://www.teradata.com/white-papers/Enabling-the-Temporal-Data-Warehouse.

[11] Snodgrass, R. 1995. TSQL2 Temporal Query Language, Springer Verlag.

[12] Snodgrass, R. 2008. A Case Study of Temporal Data, http://www.teradata.com/white-papers/A-Case-Study-of-Temporal-Data-eb6237/?type=WP

[13] Teradata, 2013. Teradata: SQL Functions, Operators and Predicates,http://tunweb.teradata.ws/tunstudent /TeradataUserManuals/SQL_Reference__Functions..pdf.

[14] Wijsen, J. 2006. Temporal Integrity Constraints, http://informatique.umons.ac.be/ssi/jef/tic.pdf

[15] Zhou, X., Wang, F. and Zaniolo, C. 2005. Efficient Temporal Coalescing Query Support in RDBMSs, LNCS **4080.**