

Temporal Data in Relational Database Systems: A Comparison

Dušan Petkovic¹

¹ University of Applied Sciences
Hochschulstr. 1, Rosenheim, 83024, Germany
petkovic@fh-rosenheim.de

Abstract. Several database systems have implemented temporal data support, partly according to the model specified in the last SQL standard and partly according to other, older temporal models. In this article we use the most important temporal concepts to investigate their implementations in enterprise database systems. Also, we discuss strengths and weaknesses of these implementations and give suggestions for future extensions.

Keywords: database systems, temporal data, temporal concepts

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 [3] or have implemented concepts according to other temporal models. Hence, we will evaluate these systems in relation to the following general concepts:

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

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, we can build the

history of all such timestamps in relation to the past and current time, but not in relation to future. Furthermore, only current values may be updated, and these 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 PK is also the temporal PK of the corresponding table.

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 valid time period of the corresponding tuple in the parent table [15].

1.3 Coalescing

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. The goal of coalescing is that the “merge” operation of the corresponding tuples should be done by the system and not by users applications.

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 [2].

1.4 PERIOD Data Type

The PERIOD data type is specified as a time interval which comprises the set of subsequent 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 main advantage of the PERIOD data type is that it can be used to represent time intervals in natural way [7].

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. A database system supporting this data type usually implements several temporal functions. The following are examples of such functions: CONTAINS, EQUALS and OVERLAPS. A proposal, how this type can be specified is given in [14]. The functions of the data type are based upon so called Allen's operators [1].

1.5 Implicit vs. Explicit Timestamps

The association of time with 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. 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 must supply valid times of facts. Therefore, updating facts in valid time and bitemporal data models must treat time explicitly and are forced to represent, how valid times of facts should be specified.

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 section discusses which concepts are supported in Oracle. The structure of this section is identical to the structure of the previous two. Section 6 describes temporal concepts in the future version of MS SQL Server. The last section summarizes the results and shows our conclusions.

2 IBM DB2

IBM DB2 supports temporal data since Version 10 [9]. 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.

2.1 IBM DB2: Time Dimensions

DB2 supports valid time as well as transaction time. The union of valid and transaction time is supported, ie. bitemporal tables are supported by DB2.

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. The CREATE (ALTER) TABLE statement is extended with the PERIOD clause, which specifies the time interval. Example 1 shows the creation of a table with valid time.

Example 1

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

Temporal columns **EStart** and **EEnd** are explicitly specified and used 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. In other words, the syntax of INSERT has not been changed. Besides the convenient syntax for the UPDATE statement, DB2 supports the FOR PORTION clause, which specifies an additional temporal condition (see Example 2).

Example 2

```
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	ENO	EDEPT	ESTART	EEND
-----	-----	-----	-----	-----	-----	-----	-----
12345	3	2013-01-01	2014-01-01	12345	4	2013-07-01	2014-01-01
12345	4	2014-01-01	9999-12-31	12345	4	2014-01-01	2014-07-01
(The content of table before				12345	3	2013-01-01	2013-07-01
UPDATE has been executed.)				12345	4	2014-07-01	9999-12-31

The tuples in Example 2 are 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 FOR PORTION clause can be used in the similar ways with DELETE.

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:

- FOR BUSINESS_TIME AS OF ...
- FOR BUSINESS_TIME BETWEEN ... AND ...
- 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.

Support of Transaction Time. IBM DB2 supports transaction time. Similarly to valid time, these tables include two columns, one for the start and the other for the end of transaction time (see Example 3). The data type of these columns has to be **TIMESTAMP**.

Example 3

```
CREATE TABLE T_Emp(ENo INT PRIMARY KEY NOT NULL,EDept INT
,sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN,
  sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
  trans_start TIMESTAMP(12) GENERATED ALWAYS AS
    TRANSACTION START ID, 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 definition of transaction time involves altogether two tables: The first one (see Example 4) 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 and name the history table

Example 4

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

The INSERT statement has the same syntax as convenient INSERT. Note that only values of time-invariant columns have to be explicitly specified, while the system implicitly inserts all time-variant values.

After execution of an UPDATE statement, the non-current part of each tuple is moved from the current to the corresponding history table. (The same is true for the DELETE statement.)

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, which are identical to the same options for the valid time. (The only difference is that instead of phrase “BUSINESS_TIME”, “SYSTEM_TIME” is used.

Bitemporal Tables. A bitemporal table is a union of tables with valid and transaction time (see Example 5).

Example 5

```
CREATE TABLE BI_Emp(ENo INTEGER 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));
```

2.2 IBM DB2: Temporal Key Constraints and Coalescing

IBM DB2 supports the WITHOUT OVERLAPS clause in the PRIMARY KEY option of the CREATE (ALTER) TABLE statement [4]. This clause ensures that each value of the relational primary key is unique at any given point in valid time (see Example 1). The system does not support coalescing.

2.3 IBM DB2: Implicit vs Explicit Timestamps and PERIOD Type

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. In that case, “SELECT * FROM table_name” does not display values of time-variant columns. These values must be explicitly named in to be displayed (see Example 14). DB2 does not support the PERIOD type.

3 Teradata

Teradata supports temporal data since Version 10. Teradata’s temporal model is based upon the TSQL2 model [13] and contains three general features: time dimensions, the PERIOD data type and temporal qualifiers.

3.1 Teradata: Time Dimensions

Support of Valid Time. Time-variant attributes in Teradata’s valid time tables are specified using the PERIOD data type. An attribute of the PERIOD type can be DATE or TIMESTAMP (see Example 6).

Example 6

```
CREATE MULTISET TABLE Emp( ENo INTEGER NOT NULL, EDept INTEGER NOT NULL,
    Emp_period PERIOD(DATE) NOT NULL AS VALIDTIME) PRIMARY INDEX (ENo);
```

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 one of four temporal qualifiers, which are used for specification of conditions in relation to time-variant columns:

- a) CURRENT
- b) AS OF
- c) SEQUENCED
- d) NONSEQUENCED

All qualifiers can be specified with the VALIDTIME keyword, and with the TRANSACTION keyword. The CURRENT qualifier selects only the current tuples, i.e. tuples with valid time values related to the current 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. 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. NONSEQUENCED does not have any logical equivalent in DB2.

Example 7

SEQUENCED VALIDTIME

INSERT INTO Emp (Eno, EDept, Emp_period)

VALUES(12345, 4, PERIOD(DATE '2014-01-01', UNTIL_CHANGED));

The INSERT statement in Example 7 contains values of the DATE type, which define the start and end of valid time. For this reason each INSERT statement must be prefixed with the SEQUENCED VALIDTIME clause. Teradata supports the special value „UNTIL_CHANGED“ for the „forever“ value. The UPDATE statement in Example 8 is semantically equivalent to UPDATE in Example 2. The DELETE statement can be specified in the same way.

Example 8

SEQUENCED VALIDTIME PERIOD (DATE '2013-07-01', DATE '2014-07-01')

UPDATE Emp SET Edept = 4 WHERE Eno = 12345;

Support of Transaction Time. The TRANSACTIONTIME clause defines a time period as transaction time (see Example 9). The transaction time-variant column must be of the TIMESTAMP WITH TIME ZONE type.

Example 9

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

Teradata also supports temporal qualifiers for transaction time. All qualifiers described earlier can be used for transaction time (with TRANSACTION keyword).

The syntax of INSERT and UPDATE statements in Teradata is identical to the syntax of the same statements in IBM DB2. Teradata supports temporal extensions for SELECT, semantically similar to those extensions defined in IBM DB2:

- 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 (see Example 10).

Example 10

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 would semantically correspond to the DB2's FOR SYSTEM_TIME FROM ... TO ... clause. SEQUENCED TRANSACTIONTIME can be implemented using the NONSEQUENCED TRANSACTIONTIME clause and the OVERLAPS operator [12]. Teradata supports bitemporal tables, too.

3.2 Teradata: Temporal Key Constraints and Coalescing

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 first 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. The second constraint ensures that the value for the constrained column in a tuple is unique for all instances of time, including past, current, and future. The semantics of this constraint is identical to the semantics of the DB2's WITH OVERLAPS clause. The third constraint treats a time-variant column as a non-temporal column. Teradata does not support referential constraints on tables with valid time [10].

Teradata does not support actually coalescing of data. The only way to coalesce data is using the P_NORMALIZE function, but it works only for the output. The implementation of coalescing is under way and is based upon the paper [16].

3.3 Teradata: PERIOD Data Type and Implicit Timestamps

Teradata is the only DBMS which supports the PERIOD data type. The system supports temporal operators, such as CONTAINS, OVERLAPS and MEETS [11].

Teradata supports implicit timestamps. 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 temporal columns can be displayed is naming them explicitly in the projection of a query (see Example 14).

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: valid time is based upon the component called “Temporal Validity”, while “Flashback Data Archive” supports transaction time.

4.1 Oracle: Time Dimensions

Support of Valid Time. The PERIOD clause in the CREATE TABLE statement is used to specify valid time intervals (see Example 11). Whether time-variant attributes are implicitly or explicitly defined depends on the PERIOD clause [5].

Example 11

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

The first statement in example above specifies explicitly the names of the two in the PERIOD clause. The second statement creates a table with implicitly defined time-variant attributes. The names of these attributes, **valid_time_start** and **valid_time_end**, are derived from the name of the PERIOD clause.

Support of Transaction Time. The creation of a table with transaction time requires that an archive is created first [6].

Example 12

```
CREATE TABLE TEmp(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. The same is true for the UPDATE statement (see Example 2).

Concerning the 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. The second query shows how to retrieve a time interval.

Example 13

```
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');
```

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

4.2 Oracle: Temporal Constraints and Coalescing

Oracle does not support temporal key constraints. The user has to define time-variant columns for valid as well as transaction time. The system does not support any form of coalescing.

4.3 Oracle: Implicit vs. Explicit Timestamps and PERIOD Type

Oracle supports implicit as well as explicit timestamps. To specify explicit timestamps, the temporal columns have to be defined and the PERIOD clause must contain their names. If the PERIOD clause does not contain the specification of

temporal columns their names are derived from the name of this clause. (The PERIOD data type is not supported by Oracle.)

Example 14

```
SELECT * FROM Emp_1;
```

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

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

5 Microsoft SQL Server

Microsoft will support temporal data with 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 section will be different than the structure of the other sections. (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 this paper, SQL Server 2016 supports only transaction time. (Microsoft refers to the tables that implement transaction time as "temporal tables".)

Example 15

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

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.

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

- AS OF <date_time>
- FROM <start_date_time> TO <end_date_time>
- BETWEEN <start_date_time> AND <end_date_time>
- CONTAINED IN (<start_date_time> , <end_date_time>)

The first three clauses correspond to the clauses specified in the SQL standard, while the last one returns a result with the values of all old versions that were opened and closed within the specified time range defined by the two parameters in CONTAINED. (The syntax of UPDATE and DELETE has not been extended with any temporal clauses.)

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.

From our point of view, the 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. We can hope that the implementation of these concepts will happen soon.

References

- [1] Allen, J. - Maintaining knowledge about temporal intervals, in CACM, 1983.
- [2] Böhlen, M., Snodgrass, R. - Coalescing in Temporal Databases, VLDB, 1996
- [3] ISO/IEC 9075-2: Information technology—DB languages—Part 2, 2011.
- [4] Nicola, M.; Sommerlandt, M. - Managing time in DB2 with temporal consistency, www.ibm.com/developerworks/data/library/techarticle/dm-1207/index.html, 2012.
- [5] Multitemporal Features in Oracle 12c, www.salvis.com/blog/2014/01/04/multi-temporal-database-features-in-oracle-12c/
- [6] Oracle Flashback Data Archive, <http://www.oracle-dba-online.com>
- [7] Petković, D. – Modern Temporal Data Models: Strengths and Weaknesses, BDAS 2015, Springer, Communication in Computer Science, Vol. 521.
- [8] Petković, D. - SQL Server 2016, Beginner's Guide, McGraw-Hill, 2016 (to appear)
- [9] Saracco, C., Nicola, M., Gandhi, L. - A matter of time: Temporal data management in DB2, www.ibm.com/developerworks/data/library/techarticle, 2012.
- [10] Sannik, G.; Daniels, F. - Enabling the Temporal Data Warehouse, <http://www.teradata.com/white-papers/Enabling-the-Temporal-Data-Warehouse>.
- [11] Snodgrass - Developing Time-Oriented Applications in SQL, Morgan Kaufman
- [12] Snodgrass, R. - A Case Study of Temporal Data, <http://www.teradata.com/white-papers/A-Case-Study-of-Temporal-Data-eb6237/?type=WP>
- [13] Snodgrass, R. - The TSQL2 Temporal Query Language, Springer Verlag, 1995.
- [14] Teradata: SQL Functions, http://tunweb.teradata.ws/tunstudent/TeradataUserManuals/SQL_Reference_Functions_Operators_Expressions_Predicates.pdf.
- [15] Wijzen, J. – Temporal Integrity Constraints, www.informatique.umons.ac.be/jef
- [16] Kateb, M. Ghazal, A. - An Efficient SQL rewrite approach for temporal coalescing in Teradata, DEXA 2012, LNCS, Vol 7447, pp.375-383.

New Advances in Information Systems and
Technologies

Rocha, Á.; Correia, A.M.; Adeli, H.; Reis, L.P.; Mendonça
Teixeira, M. (Eds.)

2016, XXVII, 1130 p. 331 illus., 230 illus. in color.,
Softcover

ISBN: 978-3-319-31231-6