Valid Time Database Implementation Using Oracle11g Workspace Manager

Sitti Rugtanom and Suphamit Chittayasothorn
Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology, Ladkrabang, Bangkok 10520 THAILAND

Abstract

Valid time database comprises valid time state tables, which keep facts that are currently true, believed to be true in the past, and expected to be true in the future. Temporal queries on these tables are both useful and necessary for many applications. However, the introduction of valid time periods to database tables leads to complex data manipulations especially delete and update operations. Conventional Database Management Systems users have to use standard SQL to handle temporal queries and temporal data manipulations which lead to long and complicated SQL statements. This paper investigates the valid time temporal database features which are available in Oracle11g. We found that the Oracle11g Workspace Manager provides valid time temporal database operations comparable to those appear in temporal database literatures.

Keywords: Temporal database, Valid Time State Table, Oracle11g Workspace Manager

1. Introduction

Temporal databases are databases that support time validity of facts and also the time that the facts are current in the database. The time validity is called the valid time which is the time that the fact is true according to current belief. Unlike conventional databases, that keep only current facts, databases with the valid time support also keep facts which were true but may no longer be true, facts which are currently true, and also facts which are expected to be true in the future. This kind of databases is for time varying information and the database is referred to as valid time database and their tables are called valid time state tables [1].

There is another kind of temporal database which keep the time that the fact is current in the database. The fact itself may be time varying or non time varying. This kind of temporal database is called transaction time database. In the case that the fact is not time varying, with respect to the human lifespan, the non time varying fact usually refers to measurements and such measurements can by more precisely obtained as time passes and measuring technology progressed. Classic examples include the brightness of stars, the height of mountain peaks etc. They are believed to be fixed but can also be corrected for better precision. Transaction time database keeps track of these changes. Databases for time varying information which has both valid time and transaction time are referred to as bitemporal databases and their corresponding tables are called bitemporal tables. In this paper, we concentrate only on the valid time database. We present how Oracle11g workspace manager handle valid time state tables. The creation of such tables, the insertion, deletion and updating with time consideration together with the temporal queries are presented.

2. Valid Time State Table

Databases keep facts and facts can be changes over time. Insertion, deletion, and modification are common operations on databases. In conventional databases, only current facts are available. Deleted facts are physically removed from the database and updated facts have old values physically replaced by new ones. Historical information is not available unless deliberately kept with timestamps attached. Such introduction of time into databases was once believed to be straight forward and trivial. However, the introduction of time into databases which is technically called temporal databases is now recognized as a research area. The database table which keeps different states of objects or relationships at different time period is now referred to as a valid time state table.

Even though the introduction of time to databases in the research community is well known to database researchers [1],[2],[3] practitioners find difficulties implementing them on commercially available DBMSs. This is due to the fact that the temporal database features are not parts of the SQL standard currently implemented by them. Richard Snodgrass, a top researcher in this area, tried to introduce the temporal database concepts and the importance of temporal applications to the public by writing articles on the topic and published them in *Database Programming and Design* during June – October 1988. The collection of these articles is republished as [4]. The paper clearly demonstrates the difficulty of temporal data handling in relational databases and proposed an extended SQL syntax which reduces the complication of the temporal SQL queries. His book on the development of temporal database applications was published in year 2000 [5].

In this paper, we present the use of Oracle11g workspace manager [6] to handle temporal database operations. A simple table POSITIONS which keeps the employee id and position at work of employees for demonstration purpose. The table can be created using a simple SQL statement:

```
CREATE TABLE  positions (
  emp_id        NUMBER PRIMARY KEY,
  job_position  VARCHAR2(30) );
```

This is a non temporal table which has emp_id as its primary key. This means that the value of emp_id must be both unique and not null. However, the temporal version of the table, as shown in Figure 1, where valid time from_date and to_date are introduced causes the emp_id values to appear more than once and violates the primary key constraint. In fact, the primary key of this temporal table is (emp_id,from_date). An immediate consequence is the more complicated referential integrity declarations since the foreign keys will need to refer to the from_date as well. To avoid this complicated situation, we refer to the non temporal primary key as the *apparent primary key*. The value of this apparent primary key must be unique at any point in time, including the current time, of course. It is the user's responsibility to specify this apparent primary key at the table creation time.

EMP_ID	JOB_POSITION	FROM_DATE	TO_DATE
46	Manager	2005-02-16	9999-12-31
47	Manager	2012-06-01	9999-12-31
84	Project Manager	2008-01-01	2010-01-01
85	Senior Engineer	2010-01-01	2012-01-01
101	Programmer	2010-01-01	2012-01-01
156	Webmaster	2008-01-01	2010-01-01
77	Manager	2005-05-01	2007-01-01
77	Manager	2008-01-01	9999-12-31
78	Programmer	2006-01-01	2006-11-01
79	Programmer	2006-06-01	9999-12-31
84	System Analyst	2005-02-16	2008-01-01
84	System Analyst	2010-01-01	9999-12-31
85	Engineer	2007-01-20	2010-01-01
101	Senior Programmer	2012-01-01	9999-12-31

Figure 1. A Valid time state table POSITIONS

The valid time state table in Figure 1 presents facts and the valid time period which shows when the facts are true. Facts which are no longer true is also available. Using the Oracle 11g workspace manager one may simply introduce the valid time to a table and make it a Validtime state table. The command is:

```
EXECUTE  DBMS_WM.EnableVersioning( 'positions', 'VIEW_WO_OVERWRITE',
FALSE,  TRUE);
```

The last two parameters inform the DBMS if the table is a spatial table (in this case it is FALSE) and valid time (in this case it is TRUE). The system will automatically add the valid time periods from_date and to_date and handle temporal database operations automatically as well. The users simply specify the valid time period which is to be applied to the database. Temporal database operations are presented in the following sections.

The valid time format is the so called closed-open format. The from_date is the date when the fact is true. However, the to_date is the first date that the fact is false. For example, from_date 2005-05-01 and to_date 2007-01-01 means that the fact is valid from May 1, 2005 to December 31, 2005. This close-open format is well accepted because it is convenience to check period continuity. The from_date of a row can be matched with the to_date of the past row. Note that the date December 31, 9999 is the maximum value of the date data type and represents infinity. The infinity TO_DATE value means the fact is still current. This notation of current validity is well accepted by the academic world as well as in practice. However, C. J. Date [3] expresses concerns on this future date and proposes that a moving NOW value should be introduced. Oracle allows null values in the to_date and interprets them as "still valid". When a row is inserted, the null value is specified as "until changed".

There are also concerns on the redundancy issue. Apart from redundant values in the normal database sense, valid time state tables are prone to have redundancies which have same information values over the same period of time. These are called *sequence redundancy*. This kind of redundancy could take place in the past, at present, or in the future. They have the same fact values (value equivalent) during the same period of time. A special case of the sequence redundancy is the *current redundancy* where fact values are redundant at the current point of time. These redundancies can be handled by the database management system (DBMS) using user-supplied trigger codes or stored procedures.

3. Temporal Queries

Queries on the valid time state table can also be classified into three categories; namely current, sequence and non sequence. The current query retrieves facts which are true at present. The sequence query retrieves facts which are true during a given period of time. The nonsequence query retrieves facts on the time attributes. It treats the valid time attributes the same as other non temporal attributes.

3.1 Current Queries

Current queries are queries that expect current results; facts which are currently true and satisfy the search conditions. In principle, current queries (and other current operations) on temporal database system should be compatible with conventional databases. Current queries are therefore almost identical to normal SQL queries except the references to valid time columns. As an example, suppose we would like to know *the position of all currently employed employees*. A simple, unqualified SQL select statement will work provided that the current valid time setting is first issued. The SetValidTime keyword without parameters means current valid time setting. The commands and corresponding result are shown in Figure 2. Note that the "current" condition does not have to be specified in the query and the result is actually the current subset of Figure 1.

```
EXECUTE  DBMS_WM.SetValidTime( );

SELECT    emp_id,  job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL, 'YYYY-MM-DD')       TO_DATE
FROM  positions  e;
```

```
EMP_ID  JOB_POSITION                 FROM_DATE     TO_DATE
------  ---------------------------  ------------  ----------
46      Manager                      2005-02-16    9999-12-31
47      Manager                      2012-06-01    9999-12-31
77      Manager                      2008-01-01    9999-12-31
79      Programmer                   2006-06-01    9999-12-31
84      System Analyst               2010-01-01    9999-12-31
101     Senior Programmer            2012-01-01    9999-12-31
```

Figure 2. A current query and result

3.2 Sequence queries

As mentioned earlier, "sequence" means temporal consideration. Queries on past, present and future facts are feasible. Without special temporal features, long and complex SQL statements are often required for queries which are common in practice and sound very simple. However, they are not easy to formulate in conventional SQL. For example, if we want to know *who were in the same position during the same period of time?* Suppose a person was a manager during the period p1 and another person was a manager during a period p2, there are four possible cases that the query must cover; namely, p1 begins before p2, p2 begins before p1, p1 contains p2, and p2 contains p1. Using standard SQL, there need to be four SQL queries; one for each case and the final result is the union of their results.

Using Oracle11g Workspace manager the WM_OVERLAPS keyword covers all the four cases and the apparently complicated query becomes a straight forward one as shown in Figure 3.

```
EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME ,
DBMS_WM.MAX_TIME );

SELECT e1.emp_id EMP_1,  e2.emp_id EMP_2,  e1.job_position,
  TO_CHAR(e2.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e2.WM_VALID.VALIDTILL, 'YYYY-MM-DD') TO_DATE
FROM    positions e1,  positions e2
WHERE  e1.emp_id      <  e2.emp_id
AND      e1.job_position =  e2.job_position
AND      WM_OVERLAPS(e1.WM_VALID, e2.WM_VALID)=1;
```

```
EMP_1     EMP_2     JOB_POSITION       FROM_DATE        TO_DATE
-----------  -----------  ---------------------  ------------------  -------------
46          47          Manager            2012-06-01       9999-12-31
79          101         Programmer         2010-01-01       2012-01-01
46          77          Manager            2005-05-01       2007-01-01
47          77          Manager            2008-01-01       9999-12-31
46          77          Manager            2008-01-01       9999-12-31
78          79          Programmer         2006-01-01       9999-12-31
```

Figure 3. A sequence temporal join query using Oracle11g Workspace Manager

Notice that the Validtime is set to cover the entire possible period of time supported by the database system. In fact, the valid time can be set to narrow down the search to a given period of time. The following example in Figure 4 shows *all employees who were employed during 2011-08-01 and 2012-04-30.* The period to which the query is applied is referred to as the period of applicability.

```
EXECUTE    DBMS_WM.SetValidTime(TO_DATE('2011-08-01','YYYY-MM-
DD'), TO_DATE('2012-04-30','YYYY-MM-DD'));

SELECT emp_id, job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
   TO_CHAR(e.WM_VALID.VALIDTILL,'YYYY-MM-DD') TO_DATE
FROM  positions e;
```

Figure 4. Valid time setting to limit the period of applicability

However, if we would like to have only *the employees whose work period is precisely equal to the required period of applicability,* a search condition with the keyword WM_EQUALS must be employed. Here the period is from 2011-08-01 to 2012-04-30. An example is shown in Figure 5.

```
EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
   TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD') TO_DATE
FROM    positions e
WHERE  WM_EQUALS(e.WM_VALID,
          WM_PERIOD(TO_DATE('2011-08-01','YYYY-MM-DD'),
                          TO_DATE('2012-04-30','YYYY-MM-DD')))=1;
```

Figure 5. The identical period searching using WM_EQUALS

Another practically common query, which is relatively hard to formulate using standard SQL even with procedural extension such as PL/SQL[7], is the overlapped case with a known period. An example is shown in Figure 6. The query is to *find employees who work as programmers when the employee number 46 is a manager.*

```
EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT  e1.emp_id  EMP_1,  e2.emp_id  EMP_2,  e2.job_position,
   TO_CHAR(e2.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
   TO_CHAR(e2.WM_VALID.VALIDTILL,   'YYYY-MM-DD') TO_DATE
FROM      positions e1, positions e2
WHERE  e1.emp_id  < e2.emp_id
AND        e1.emp_id  = 46
AND        e1.job_position = 'Manager'
AND        e2.job_position = 'Programmer'
AND        WM_OVERLAPS(e2.WM_VALID,e1.WM_VALID)=1;
```

```
EMP_1     EMP_2     JOB_POSITION        FROM_DATE       TO_DATE
------------  -----------  ----------------------  -------------------  ----------------
46          101         Programmer          2010-01-01      2012-01-01
46          78          Programmer          2006-01-01      2006-11-01
46          79          Programmer          2006-06-01      9999-12-31
```

Figure 6. A sequence query with overlapped time period checking

4. Temporal Data Manipulations

Temporal queries are very useful and the temporal support provided by Oracle 11g Workspace Manager is handy. Even though it takes less time and effort to formulate the temporal queries, one still may not be convinced that the temporal support provided is significant. In this section, the more difficult parts of temporal database operations, the data manipulations, especially delete and update operations are discussed.

Temporal insertion is straight forward. The period of validity must be supplied first before the insertions otherwise they will be considered as current validity. Delete and update are complicated. The period of applicability of the operation and the period of validity of existing facts must be considered together. A temporal delete (or logical delete) may result in several physical updates and a physical insert. A temporal update (or logical update) may result in a physical update and two physical inserts. Note that the term "logical" here refers to the temporal database level where the temporal applications see the database. The term "physical" refers to the conventional database as seen by non-temporal applications. In the database area, these terms are not fixed to a level of abstraction but are relative to the level of operations which is being referred to.

4.1 Sequence Insert

Sequence insertion is the most straight forward activity. One simply specify the period of fact validity. An example, Figure 7 shows the fact that employee #44 was a system analyst during the period 2005-02-16 and 2008-01-01 is inserted. Note that this is the close-open format which means that the fact does not hold on the day 2008-01-01. A current insert is simply a special case of the sequence one. In this case the from_date is SYSDATE and the to_date is DBMS_WM.MAX_TIME (9999-12-31) or DBMS_WM.UNTIL_CHANGED.

```
INSERT INTO positions VALUES(44, 'System Analyst',
     WMSYS.WM_PERIOD(TO_DATE('2005-02-16','YYYY-MM-DD'),
                               TO_DATE('2008-01-01','YYYY-MM-DD')));

INSERT INTO positions VALUES(47, 'Manager',
    WMSYS.WM_PERIOD(SYSDATE,DBMS_WM.UNTIL_CHANGED));
```

Figure 7. A sequence and a current insert

4.2 Sequence Delete

Sequence deletion is a challenging task. A fact which is valid during a period of time (this is referred to as the period of validity: PV) is to be removed (deleted) from the database only for another period of time (this is referred to as the period of applicability: PA). There are four possible cases that the PA and PV are overlapped. Similar to the sequence queries where a single OVERLAP operator takes care of all the four cases, the sequence deletion features of Oracle11g Workspace Manager takes care of all the four possible cases automatically. The user simply supplies the PA; the DBMS will check and decided which of the four cases is applicable. The four temporal deletion cases are shown in Figure 8.

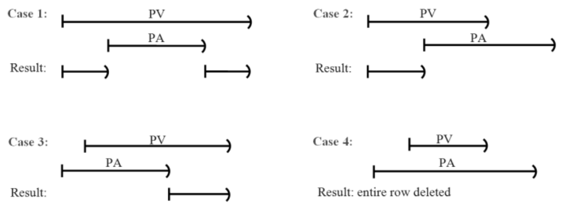


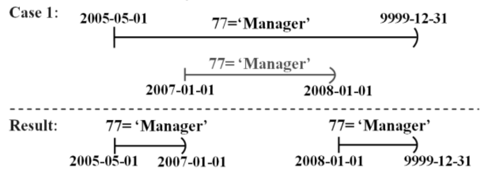
Figure 8 Sequenced Deletion Cases [4]

In the case 1, PV contains PA. This means that the fact which is true during PA is to be removed from the database. The physical operations which are equivalent to the temporal deletion in this case include one update and one insert. The update sets the to_date of the fact row to be the from_date of the PA. The insert inserts a new row with the old fact value with the from_date equals to the to_date of PA and to_date equals to the to_date of PV.

In the case 2, PV overlaps PA in such a way that PV begins before PA. The physical operation is only one update. The to_date of the fact row is simply set to be the from_date of PA. In the case 3, PV also overlaps PA. However, in this case, PA begins before PV and the physical operation is also only one update. The from_date of the fact row is set to the to_date of PA. The final case 4 is the case where PA contains PV and the entire fact row has to be physically deleted.

When a sequence delete with a PA is applied to the database, the issuer may not know the PV of the fact rows which are to be deleted. The deletion request therefore has to cover all the four possible cases. In conventional SQL, this includes an insert, some updates and a delete. Using Oracle11g Workspace Manager, the user simply supplies the period of applicability PA and a simple delete statement. All the four cases are taken care of automatically.

Figure 9 demonstrates the deletion case 1 using Oracle11g Workspace Manager syntax. We first insert a new demonstration row which states the fact that employee 77 is a manager from 2005-05-01 until now (9999-12-31). A simple selection follows the insertion to show that the row is inserted. After that, the period of applicability is set to be 2007-01-01 and 2008-01-01; follows by a simple delete statement. This is the sequence delete. The last SQL select statement shows the two-row results are according to the case 1.



```
EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD') TO_DATE
FROM  positions e
WHERE emp_id=77;
```

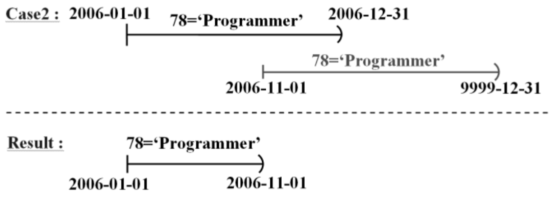
```
EMP_ID   JOB_POSITION              FROM_DATE        TO_DATE
-------- ------------------------- ---------------- ------------
77       Manager                   2005-05-01       9999-12-31
```

```
EXECUTE   DBMS_WM.SetValidTime(TO_DATE('2007-01-01','YYYY-MM-
DD'),TO_DATE('2008-01-01','YYYY-MM-DD'));

DELETE FROM positions WHERE emp_id=77;

EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
   TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD') TO_DATE
FROM   positions e
WHERE emp_id=77;
```

```
EMP_ID   JOB_POSITION              FROM_DATE        TO_DATE
-------- ------------------------- ---------------- ------------
77       Manager                   2005-05-01       2007-01-01
77       Manager                   2008-01-01       9999-12-31
```

Figure 9 Oracle SQL codes for Case 1 Sequence Deletion

Figure 10 demonstrates the deletion case 2 using Oracle11g Workspace Manager syntax. We first insert a new demonstration row which states the fact that employee 78 is a Programmer from 2006-01-01 until 2006-12-31. A simple selection follows the insertion to show that the row is inserted. After that, the period of applicability is set to be 2006-11-01 and 9999-12-31; follows by a simple delete statement. This is the sequence delete. The last SQL select statement shows the result row which is according to the case 2.



```
INSERT INTO positions VALUES( 78, 'Programmer',
    WMSYS.WM_PERIOD(TO_DATE('2006-01-01','YYYY-MM-DD'),
                    TO_DATE('2006-12-31','YYYY-MM-DD')));

EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id,  job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
   TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD') TO_DATE
FROM    positions e
WHERE  emp_id=78;
```

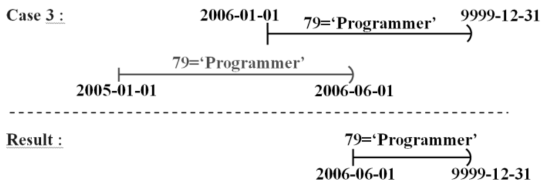
```
EMP_ID   JOB_POSITION              FROM_DATE        TO_DATE
-------- ------------------------- ---------------- -------------
78       Programmer                2006-01-01       2006-12-31
```

```
EXECUTE   DBMS_WM.SetValidTime(TO_DATE('2006-11-01','YYYY-MM-
DD'), DBMS_WM.MAX_TIME);

DELETE FROM positions WHERE emp_id=78;

EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD') TO_DATE
FROM  positions e
WHERE emp_id=78;
```

```
EMP_ID   JOB_POSITION                 FROM_DATE           TO_DATE
-----------  -----------------------------  -------------------  ---------------
78          Programmer                   2006-01-01          2006-11-01
```

Figure 10 Oracle SQL codes for Case 2 Sequence Deletion

Figure 11 demonstrates the deletion case 3 using Oracle11g Workspace Manager syntax. We first insert a new demonstration row which states the fact that employee 79 is a Programmer from 2006-01-01 until now (9999-12-31). A simple selection follows the insertion to show that the row is inserted. After that, the period of applicability is set to be 2005-01-01 and 2006-06-01; follows by a simple delete statement. This is the sequence delete. The last SQL select statement shows the result row which is according to the case 3.



```
INSERT  INTO positions VALUES( 79, 'Programmer',
    WMSYS.WM_PERIOD(TO_DATE('2006-01-01','YYYY-MM-DD'),
                              DBMS_WM.MAX_TIME));

EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD')  FROM_DATE,
   TO_CHAR(e.WM_VALID.VALIDTILL,   'YYYY-MM-DD')  TO_DATE
FROM     positions e
WHERE emp_id=79;
```

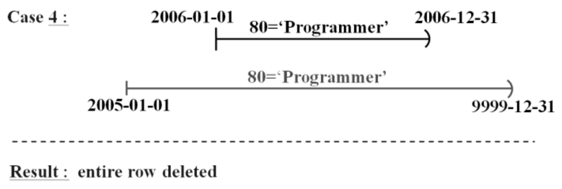
```
EMP_ID   JOB_POSITION                 FROM_DATE           TO_DATE
-----------  -----------------------------  -------------------  ---------------
79          Programmer                   2006-01-01          9999-12-31
```

```
EXECUTE DBMS_WM.SetValidTime(TO_DATE('2005-01-01','YYYY-MM-
DD'),TO_DATE('2006-06-01','YYYY-MM-DD'));

DELETE FROM positions WHERE emp_id=79;

EXECUTE   DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
   TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL,   'YYYY-MM-DD') TO_DATE
FROM  positions e
WHERE emp_id=79;
```

```
EMP_ID   JOB_POSITION                 FROM_DATE           TO_DATE
-----------  -----------------------------  -------------------  ---------------
79          Programmer                   2006-06-01          9999-12-31
```

Figure 11 Oracle SQL codes for Case 3 Sequence Deletion

Figure 12 demonstrates the deletion case 4 using Oracle11g Workspace Manager syntax. This is the case where PA contains PV and the entire fact row will be deleted. We first insert a new demonstration row which states the fact that employee 80 is a Programmer from 2006-01-01 until now (2006-12-31). A simple selection follows the insertion to show that the row is inserted. After that, the period of applicability is set to be 2005-01-01 and 9999-12-31; follows by a simple delete statement. This is the sequence delete. The last SQL select statement shows no result row which is according to the case 4.



```
INSERT  INTO  positions VALUES( 80,  'Programmer',
   WMSYS.WM_PERIOD(TO_DATE('2006-01-01','YYYY-MM-DD'),
                          TO_DATE('2006-12-31','YYYY-MM-DD')));

EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT  emp_id,  job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD')  FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL,   'YYYY-MM-DD') TO_DATE
FROM     positions  e
WHERE emp_id=80;
```

```
EMP_ID   JOB_POSITION                 FROM_DATE           TO_DATE
-----------  -----------------------------  -------------------  ---------------
80          Programmer                   2006-01-01          2006-12-31
```

```
EXECUTE     DBMS_WM.SetValidTime(TO_DATE('2005-01-01','YYYY-
MM-DD'), DBMS_WM.MAX_TIME);

DELETE  FROM  positions WHERE emp_id=80;

EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id,  job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD')  FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL,   'YYYY-MM-DD') TO_DATE
FROM    positions e
WHERE emp_id=80;
```

```
EMP_ID   JOB_POSITION                 FROM_DATE           TO_DATE
-----------  -----------------------------  -------------------  ---------------


0 rows selected
```

Figure 12 Oracle SQL codes for Case 4 Sequence Deletion

4.3 Sequence Update

Sequence update is also a challenging task. A fact which is valid during the period of validity PV is to be modified during the period of applicability PA. Similar to the sequence deletions, there are four possible cases that the PA and PV are overlapped. They are illustrated in Figure 13. Here the sequence update features of Oracle11g Workspace Manager also take care of all the four possible cases automatically. The user simply supplies the PA; the DBMS will check and decided which of the four cases are applicable.

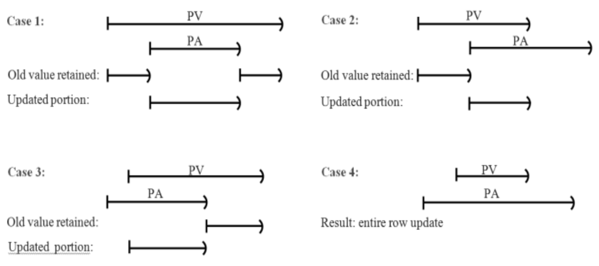
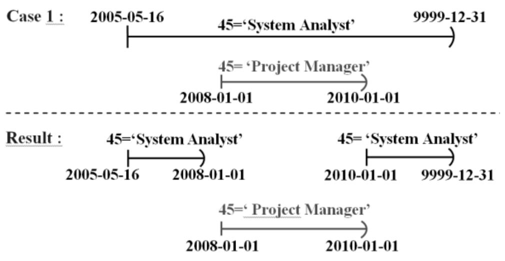


Figure 13 Sequenced update cases [4]

The case 1 is when the PV contains PA. It is an update from the old value to the new value during the PA period; the result comprises three parts. The first part is the original row with the old value retained. Its from_date is the same as the from_date of the PV and the to_date is equal to the from_date of PA. The second part is the newly updated value. This is a new row whose from_date and the to_date is precisely the same as the PA. The third part is the row which has the old value. Its from_date is the to_date of the PA and the to_date is the to_date of the PV. This third part is also a newly inserted row. In summary, an update in the case 1 physically requires one update and two insert operations.

The case 2 is when PV precedes and overlaps PA. In this case, the old value in the original row is retained until the beginning of PA. Its to_date is set to the from_date of PA. A new row with the new value is inserted. Its from_date is the from_date of PA and the to_date is the to_date of PV. One update and one insert are required in total for this case. Similarly, the case 3 requires one update and one insert. The case 4 requires the entire existing row to be updated since PA contains PV.

Figure 14 demonstrates the update case 1 using Oracle11g Workspace Manager syntax. We first insert a new row which states the fact that employee 45 is a system analyst from 2005-05-16 until now (9999-12-31). A simple selection follows the insertion to show that the row is inserted. After that, the period of applicability is set to be 2008-01-01 and 2010-01-01; follows by a simple delete statement. This is the sequence delete. The last SQL select statement shows the three-row results according to the sequence update case 1. Due to space limitation, we can only demonstrate this case. The other cases are according to the sequence update principle.



```
INSERT INTO positions VALUES( 45, 'System Analyst',
   WMSYS.WM_PERIOD(TO_DATE('2005-05-16','YYYY-MM-DD'),
                             DBMS_WM.MAX_TIME));

EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT emp_id, job_position,
  TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD') FROM_DATE,
  TO_CHAR(e.WM_VALID.VALIDTILL, 'YYYY-MM-DD') TO_DATE
FROM    positions e
WHERE emp_id=45;
```

```
EMP_ID   JOB_POSITION             FROM_DATE     TO_DATE
-------- ------------------------ ------------- -----------
45       System Analyst           2005-05-16    9999-12-31
```

```
EXECUTE    DBMS_WM.SetValidTime(TO_DATE('2008-01-01','YYYY-
MM-DD'), TO_DATE('2010-01-01','YYYY-MM-DD'));

UPDATE positions SET job_position='Project Manager'
WHERE emp_id=45;

EXECUTE    DBMS_WM.SetValidTime(DBMS_WM.MIN_TIME,
DBMS_WM.MAX_TIME);

SELECT  emp_id, job_position,
 TO_CHAR(e.WM_VALID.VALIDFROM,'YYYY-MM-DD')  FROM_DATE,
 TO_CHAR(e.WM_VALID.VALIDTILL,  'YYYY-MM-DD')  TO_DATE
FROM    positions e
WHERE emp_id=45;
```

```
EMP_ID   JOB_POSITION             FROM_DATE     TO_DATE
-------- ------------------------ ------------- -----------
45       Project Manager          2008-01-01    2010-01-01
45       System Analyst           2005-05-16    2008-01-01
45       System Analyst           2010-01-01    9999-12-31
```

Figure 14 Sequenced update case 1

5. Conclusion

Oracle11g is a commercially available DBMS that supports temporal database operations. Facilities for the creation, querying and data manipulations of Oracle11g Workspace Manager have been tested and tried. Valid time temporal database operations which are complicated and require technical knowledge beyond conventional database practices becomes simpler tasks. However, to fully exploit the temporal database features of this DBMS require technical backgrounds and worked examples. This paper is an attempt to introduce these temporal database features and their commercially available implementation to the public.

6. References

[1] Abdullah Tansel, James Clifford, Shashi Gadia, Sushil Jajodia, Arie Segev, and Richard T. Snodgrass, **Temporal Databases: Theory, Design, and Implementation,** Benjamin Cummings Pub., 1993.

[2] R. T. Snodgrass (editor), I. Ahn, G. Ariav, D. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Kaefer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo and S.M. Sripada, **The Temporal Query Language TSQL2**, Kluwer Academic Publishers, 1995.

[3] C. J. Date, Hugh Darwen, Nikos A. Lorentzos, **Temporal Data and The Relational Model**, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2003.

[4] Richard T. Snodgrass, **Managing Temporal Data A Five-Part Series**, A Time Center Technical Report n: September 3, 1998.

[5] Richard Snodgrass, **Developing Time-Oriented Database Applications in SQL**, Morgan Kauffmann Series in Data Management, San Francisco, California, 2000.

[6] Chuck Murray, **Oracle® Database Workspace Manager Developer's Guide, 11 g Release 2 (11.2) E11826-02**, October 2009.

[7] Sheila Moore, **Oracle® Database PL/SQL Language Reference, 11 g Release 1 (11.1) B28370-05**, August 2009.