A Temporal Data Model and a Query Language Based on the OO data model

Yongmoo Suh*

ABSTRACT

There have been lots of research on temporal data management for the past two decades. Most of them are based on some logical data model, especially on the relational data model, although there are some conceptual data models which are independent of logical data models. Also, many properties or issues regarding temporal data models and temporal query languages have been studied. But some of them were shown to be incompatible, which means there could not be a complete temporal data model, satisfying all the desired properties at the same time. Many modeling issues discussed in the papers, do not have to be done so, if they take object-oriented data model as a base model. Therefore, this paper proposes a temporal data model, which is based on the object-oriented data model, mainly discussing the most essential issues that are common to many temporal data models. Our new temporal data model and query language will be illustrated with a small database, created by a set of sample transactions.

1. Introduction

There are many applications, in which it is important to capture and utilize the history of values of objects, such as banking, inventory control, personnel management, analysis of serial scientific experimental data, trend analysis in DSS, etc. But most of the conventional DBMSs do not support the temporal aspects of data. When old values are replaced with new ones, they are not retained in the database any more. As such, one cannot issue historical queries, asking for old values of objects. Also, in engineering fields such as CAD/CAM and CASE, it is frequent to make a new version of an artifact. Therefore, it is necessary to maintain the history of versions of versionable objects, so that one can figure out which one is a version of another and which version is derived from some generic

* Department of Business Administration, Korea University

object. Obviously, version management of this kind is another area that requires the support for the temporal management of data.

More than 40 temporal data models have been introduced in the literature, most of them with their own temporal query languages [15]. They discussed many issues such as types of time-line (continuous, discrete, linear, branching), timestamping represented either as time point or as time interval, unit of timestamping (attribute timestamping, tuple timestamping, table timestamping, etc), types of time to support (valid time, transaction time), 1NF vs Non-1NF, temporal normalization, temporal ordering, interpolation, homogeneity, temporal optimization, reducibility to snapshot database and so on. Most of the temporal data models are based on some logical data model such as relational model and object-oriented model. Although there are some temporal data models which are independent of the logical data models [10, 21], they eventually have to be mapped to a logical data model.

This paper proposes a temporal data model, which are based on the object-oriented data model, mainly discussing the most essential issues that are common to many temporal data models. There are some reasons for taking the object-oriented data model as a basis. First, we should note that many of the issues mentioned above stem from the fact that their models are based on the relational model. Otherwise, they do not have to be discussed. Second, time-varying attributes whose values are varying over time cannot satisfy the relational requirement of first normal form (1NF), since the value of a time-varying attribute is complex (e.g., it is basically set-valued). Similar argument that the relational data model is inappropriate as a logical data model to represent time-varying information can be found in [1]. Therefore, this paper proposes a new temporal data model based on the object-oriented data model.

This paper is organized as follows. Section 2 recapitulates the most essential issues that were commonly discussed in many existing temporal data models introduced in the literature. Section 3 starts with a set of sample transactions, to give a brief explanation why the relational data model is not appropriate as a base model to be extended for the manipulation of the time-varying information. Then, the section introduces our new temporal object-oriented data model, using the same set of sample transactions. Section 4 illustrates new operators that could be invoked in queries when retrieving information from a temporal database. They are then followed by the syntax of a new temporal SQL statement and a set of typical queries against a sample temporal database, which are assumed to be created from the set of sample transactions. Related works are introduced in section 5, and this paper ends with some concluding remarks in section 6.

2. Fundamental Issues Regarding Temporal Data Model

There have been lots of research for the past two decades on the temporal database, mainly

regarding how to represent the concept of time and regarding what operators to define that could be used in a temporal query. That is, they defined their own temporal algebra and showed that how the time-varying data can be manipulated in their temporal algebra. Though there are many desirable properties that should be satisfied by a temporal database, however, it was shown that some of the desirable properties cannot be satisfied at the same time [13]. So, in this section, we will introduce only the most essential issues among them, discussed in many research papers. Issues to be discussed here are models of time, kinds of time to support, kinds of temporal database, granularity for timestamping, and major operations to be supported by temporal database systems.

2.1 Models of time

There are two kinds of models of time: continuous model vs discrete model, and linear model vs branching model. Continuous model is to real number system, what discrete model is to integer number system. Since it is not easy to represent continuous time in discrete computer systems, most papers on temporal database adopt the discrete model [5], in which time is viewed as an infinite set of consecutive non-decomposable units, which are referred to as chronons [13]. Chronon is the shortest duration of time that is manageable by temporal database systems. Attribute values of an object (or an entity) represent a state of the object, and the state remains the same before it changes when an event occurs. This implies the state of an object lasts for some time (e.g., each state has a duration). As such, capturing the history of all the changes of a state requires to represent both the value and its valid time interval, which is a set of consecutive chronons.

Time in itself is linear. That is, for any two events, their occurring time instants can be compared, to know which one occurs earlier than the other or whether they occurred at the same time. However, in certain areas of applications, such as CAD, CASE or some engineering area, which are characterized by complex data due to hierarchical structure and interrelationships within the hierarchy, it is more important to keep information on which version of an artifact is after which other version. In this case, time is non-linear. It is rather considered as branching. In literature, some models are suggested for version control in CAD environments [4, 11] and some for version modeling and configuration management in both temporal databases and versioned databases [19, 20, 22].

2.2 Kinds of Time to Support and Kinds of Temporal Databases

There are three different kinds of time, discussed in the literature: user-defined time, valid time, and transaction time. Meaning of the user-defined time depends upon the application in which it is to be used. It is just another attribute, which can store time-related information such as birth date.

Therefore, it is not supported by most temporal query languages. The notions of valid time and transaction time are first introduced in [2] as effective time and registration time, respectively. Valid time of a fact is the time when the fact is true in the real world. Other names for the notion of valid time are real-world time, logical time, and intrinsic time. Transaction time of a fact is the time when the fact is stored in a database. Other names for this notion of transaction time are extrinsic time, physical time and registration time.

Depending upon whether valid time and/or transaction time are supported or not, databases are classified as one of the four: snapshot, rollback, historical and temporal databases. Snapshot databases do not support either valid time or transaction time. There is no way to perform retroactive changes to and historical queries against the snapshot databases. *Rollback databases* support the transaction time and historical databases support the valid time. *Temporal databases* support both valid and transaction times. Historical and temporal databases support the *historical queries* and rollback and temporal databases support the *rollback operation*. Snodgrass and Ahn [17] provides quite a good explanation of the four types of databases, by making an analogy for each case. A snapshot database is compared to the latest payroll stub, a rollback database to a collection of payroll stubs, a historical database to a resume, and a temporal database to a collection of resumes, each marked by its prepared date. They showed that in order not to lose any past information, we need to capture both the valid time and transaction time, thereby making a zero-information-loss model as is mentioned in [7]. (Note that in a rollback database, it is impossible to correct errors in the past, while in a historical database, no record of corrected errors is kept.) However, most research on temporal databases assumes that their databases support only the valid time. Historical database is updatable (retroactive or proactive changes are possible), rollback database is append-only, and temporal database is both updatable and append-only.

2.3 Granularity of Timestamping

To store the time-varying information in a database, we have to attach temporal information (e.g., timestamp) to some part of the database, e.g., a database, a relation, a tuple, an attribute, or each value of a time-varying attribute [6]. Most frequently used among them are tuple timestamping and attribute-value timestamping. Note that in the relational model, attribute-value timestamping cannot be supported, because attribute-value timestamping implies a set value for each attribute. However, attribute-value timestamping is more natural than tuple timestamping, because all time-varying attributes of a relation do not have the same change rate and some attributes may not be time-varying at all [5], and because when tuple timestamping is used, all information about a single object cannot be stored in a single tuple.

Time instant, time interval or their combination can be used as a timestamp. Clifford and Tansel [5] introduced the term *lifespan* (as an attribute to represent the duration time of a state of a tuple), which is defined as a subset of a countably infinite set of time instants. So, it can be a time instant or a time interval. They used both tuple timestamp and attribute timestamp. Clifford and Croker [3] used both tuple lifespan and attribute-value lifespan. They have represented the lifespan as a set of intervals. (They stated that representing time either as instant or as interval is a matter of convenience, but we feel that it is more than that, because depending upon the representation of time, temporal operators to be defined must be different and so must be the query language.) Lorenztos [12] represented time as an interval in his interval extended relational model IXRM. Clifford and Warren [6] have added new two attributes, STATE and EXISTS? to construct a completed relation from static relations. STATE represents a time instant and EXISTS? represents whether a certain tuple exists at a specific state. That is, they used instant timestamp. The completed relation has a tuple for each entity for every state. Historical relation is defined to be the union of the completed relations of all states. Gadia and Nair [7] used the notion of *temporal element* as value timestamp. Temporal element is a finite union of intervals. Note that the set of all temporal elements is closed under the usual set operations such as union, intersection and complement, while the set of all intervals is not. Queries in systems which use interval timestamp tend to be more complex than in other systems. Interval timestamp can also be represented using two attributes, say, START and END, as in [2].

2.4 Major Operations to be Supported

There are new operations that need to be supported by a database system that supports the manipulation of time-varying information. The classical operations for the relational data model should be extended, so that the new version of those operations can be applied to the temporal database. In other words, *select*, *project* and *join* operations should be extended in such a way that the original one and the extended one are equal in the absence of time specification. And a new version should be created for each set operation such as *union*, *intersect* and *difference*. A temporal database can be drawn as a cube, created by accumulating two-dimensional tables along the third axis of time. Operations common to most temporal data models are *time-slice* and *rollback* operations. Time-slice is the operation of slicing the cube along the time axis, according to a given time interval and rollback operation is the one, which takes the database back to the state as of some past time. One of the reasons of retaining the historical data in a temporal database is to do trend analysis against the database during a certain period of time. Therefore, there should be an operation to support this trend analysis. The concept of *moving window* corresponds to such an operation, when implemented. These

operations are invoked in the statements for data manipulation. Some of them led to new temporal SQL statements, such as TSQL[14], TOSQL[16], HSQL[18], etc.

Our temporal data model, temporal query language and operations required for our temporal data model will be explained with examples in the following sections 3 and 4.

3. A Temporal Data Model based on Object-Oriented Model

We start with an explanation why the relational data model is not appropriate as a base model to be extended to support the manipulation of time-varying information, using a set of sample transactions. Then, we explains how the object-oriented data model can be extended to support the manipulation of time-varying information. The same set of sample transactions will be used through out the paper.

3.1 Problems with Relational Data Model

As is said earlier, the relationl data model has been used most frequently as the underlying data model, to be extended for the manipulation of time-varying information. However, the relational data model does not seem to be the best choice. It is because the requirement that tables be in the first normal form in the relational data model implies that a set value or a complex value consisting of several values cannot be stored in an attribute of such tables. That is, it is not easy to store in such a table a value of a time-varying attribute, which is basically a set of complex values. To explain this, suppose we have the following set of transactions:

1) Kim started to work with salary 300, in D1 department, from Jan. 1992.
2) His salary was raised to 350, in Jan. 1993.
3) It was found in Feb. 1993, that his salary had to be raised to 330 instead of 350.
4) He moved to D2 department with a new salary 400, in July 1994.
5) His salary was raised to 500, in July 1995.

In the above series of example transactions, we have two time-varying information, salary and department of an employee. It is not easy at all to store all this time-varying information in a single table of the pure relational data model. That is, adding two attributes, *from* and *to*, to a table is not enough to store the information correctly, because we have plural time-varying attributes, and because they do not vary at the same time. As such, we have to decompose the table into two tables, *emp-salary (name, salary)* and *emp-department (name, department)*, so that each table has only one time-varying information. This decomposition results in the time-normalized tables, the concept of

which was introduced in [14]. However, even after the decomposition into time-normalized forms, we still have some problems. That is, all information about an object cannot be stored in a single tuple and join operations are required to get all information of an object. These problems can be resolved with ease, if the object-oriented data model is selected as an underlying data model. For these reasons and others, we have chosen the object-oriented data model as our base data model for supporting the various manipulation of time-varying information.

3.2 Extension of Object-Oriented Data Model

The value of time-varying attribute is varying over time, as is implied by the name. Therefore, its value is a set of (value and time-interval) pairs, where each pair represents that the value is valid only for the duration of the *time-interval*. (Note that we are using attribute timestamping to represent the valid time interval.) Time interval can be represented using two-attributes, for example, from and to, each of which is a time point. As such, we can think of two classes, *time-point* and time-interval, and a superclass of them, *time*, which can be defined as an abstract class, by collecting attributes and methods, (e.g, =, or < operations), common to the two classes, time-point and time-interval. So, we may define a *time library*, in which classes related to time are registered, together with operations that could be invoked to perform some time-related calculations.

In extending the object-oriented data model to represent time-varying information, we need the following two extensions. First, it should be possible for the database system to recognize which attributes are time-varying among the attributes that a user defines by the CREATE statement. Having recognized time-varying attributes, the database system creates a new class for each time-varying attribute. Figure 1 is the schema diagram for the above transactions, including the time

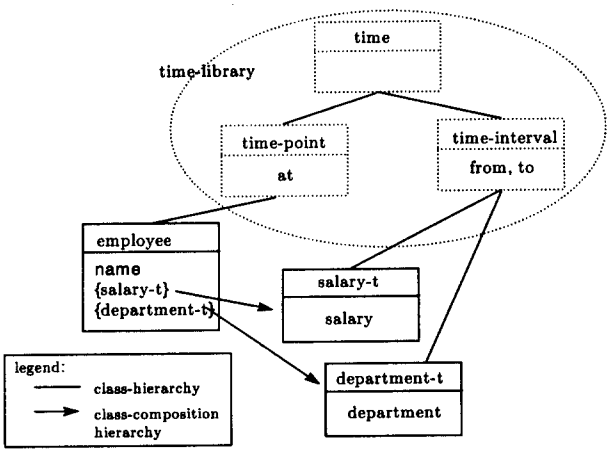

Figure 1: a sample schema diagram

library, which will be described further in section 4.1.

Attributes whose names end with '-t' in Figure 1 indicate that they are time-varying attributes. Employee class has two time-varying attributes, *salary-t* and *department-t*, from each of which a new class with the same name as the attribute name is created by the system. Those classes, representing time-varying attributes, inherit two additional attributes, from and to, from the class *time-interval* predefined in the time library, thereby having three attributes. Therefore, when each employee object is retrieved, it has three components: name, OID of salary-t and OID of department-t. Second, users should be allowed to declare whether a class is historical or temporal. In case that users want to keep all information about transactions, even about wrong transactions, they have to declare the classes related to the transactions to be temporal. Values of instances of temporal classes will be stored with two timestamps (e.g., the valid time interval and the transaction time point). If users declare classes to be historical, the valid time interval is stored to the database along with the values, when inserting instances to the historical classes. We should note that a temporal class consumes lots of storage, because every transaction causes another instance to be created for the same object. For the above example, a user has to define the employee temporal class as follows:

```
create temporal class employee as subclass of time-point
{
        name            string,
        salary-t        integer:month,
        department-t    string:day
};
```

In the above, 'salary-t integer:month' represents that the attribute salary-t is a time-varying attribute, whose value is to be stored as an integer and whose time-interval (represented as from and to, inherited from pre-defined class time-interval) is to be represented down to month (that is, month and year). All time-varying attributes are set-valued, by default. On reading the above CREATE statement, the system automatically generates two classes, salary-t and department-t as follows:

```
create historical class salary-t as subclass of time-interval
{
        salary          integer  with granularity month
};
create historical class department-t as subclass of time-interval
{
        department      string   with granularity day
};
```

Now, the above sample transactions are stored in our extended data model as follows: (Here we represented time-varying attributes as a set of triplets, instead of OIDs.)

name	salary-t	department-t	at
Kim	{(300, 01/92, now)}	{(D1, 01/92, now)}	01/01/92
Kim	{(300, 01/92, 12/92), (350, 01/93, now)}	{(D1, 01/92, now)}	01/01/93
Kim	{(300, 01/92, 12/92), (330, 01/93, now)}	{(D1, 01/92, now)}	02/01/93
Kim	{(300, 01/92, 12/92), (330, 01/93, now)}	{(D1, 01/92, 06/94), (D2, 07/94, now)}	07/01/94
Kim	{(300, 01/92, 12/92), (330, 01/93, 06/95), (500, 07/95, now)}	{(D1, 01/92, 06/94), (D2, 07/94, now)}	07/01/75

Each employee instance has a transaction time point and each time-varying attribute of the instance has a set-value of (value, valid time interval) pairs. Note that an instance has been created for each transaction and thus we do not lose any transaction information, including the second wrong transaction, which is later corrected by the third transaction. If we define the class employee as a historical class, we have only one instance, which is the same as the fifth instance in the above without transaction time attribute at, as is shown below.

name	salary-t	department-t
Kim	{(300, 01/92, 12/92), (330, 01/93, 06/95), (500, 07/95, now)}	{(D1, 01/92, 06/94), (D2, 07/94, now)}

What if there are more than one time-varying attribute that are synchronously changing over time? If we follow the above extension, we end up with as many historical classes as the number of time-varying attributes. That is not desirable. We want the system allows users to define those attributes in such a way that just one historical class is created by the system for those synchronously changing time-varying attributes. For example, suppose two attributes salary and position change always at the same time. In this case, a temporal class for these two synchronous attributes can be created as follows:

```
create temporal class employee as subclass of time-point
{
        name                    string,
        salary-t, position-t    integer:string: month
};
```

This time, the system will automatically creates a class for the two time-varying attributes as follows:

```
create historical class salary&position-t as subclass of time-interval
{
        salary                  integer,
        position                string with granularity month
};
```

It is possible to store transaction time point together with valid time interval, as is suggested by [10]. In that case, however, it takes longer time to rollback to a past time point. For this reason, we have used instance-timestamping for transaction time, while we have used attribute-timestamping for valid time interval. The granularity of the transaction time point can be adjusted, as is necessary. For example, although it is mm/dd/yy in the above, it can be changed to a much smaller granule such as mm/dd/yy hh:mm and mm/dd/yy hh:mm:ss.

4. A Temporal Query Language based on Object-Oriented Model

This section explains three things: 1) time library, 2) new SQL select statement for temporal database, and 3) queries using the new select statement.

4.1 Time Library

The basic form of the value of a time-varying attribute in our temporal data model is a set of (value, from, to) triplets. That is, in each element of the set, a value is paired with its valid time interval, which is represented by two instance variables, from and to. For the convenience of users, the following operations had better be defined.

1) operations dealing with time intervals:

① operations to deal with two time intervals such as before, after, covers, overlaps, equals, adjacent, intersection, combine, etc

② elapsedX(TI) to compute the number of days, months, or years, given a time interval, TI(X can be Days, Months, Years, etc)

③ getFrom(TI) or getTO(TI) to get the beginning or ending point from a time interval, TI

④ getTime(TVA) to extract the time interval from a time-varying attribute, TVA

⑤ getTime(O) to extract the time interval of an object O, which is computed by ORing the getTime(TVA) of all time-varying attribute TVA of the object O

⑥ getTime(B) to extract the time interval, during which the boolean expression B is true.

⑦ time slice operation, which is to slice the time dimension, given a valid time interval

Operations belonging to ①, ②, and ③ need to be defined as methods of the time-interval class of the time library, and operations belonging to ④, ⑤ and ⑥ as methods of an abstract class, which is defined as a superclass of all the temporal classes. Time slicing is defined as a new clause in our temporal SELECT statement, which is to be introduced in section 4.2. We don't think that the details of using these operations need to be explained here.

2) operations dealing with time points:

⑧ makeInterval, given two time points

⑨ operations to extract a component of a given time point (For example, getYear(TP) returns 1996 if TP is 12/25/96.)

⑩ operations to deal with now, which is a variable representing the current time point

⑪ predecessor and successor operations that return the previous time point and next time point in the linear time line, respectively (It should return different values, depending on the time granularity.)

⑫ rollback operation

Operations belonging to ⑧, ⑨, ⑩ and ⑪ needs to be defined as methods of the time-point class of the time library, and rollback operation is added as a new clause to our temporal SELECT statement.

3) operations that will be used to convert the basic form

⑬ operation that changes the basic form, {(value, from, to)} of a value of TVA, into the other form {(value, {(from, to)})}

The new form is good especially when one wants to know all the time intervals, during which a TVA has the same value. This operation is also defined as a method of the abstract class, which is defined as a superclass of all the temporal classes.

4) operations dealing with sets:

⑭ operations that deal with sets of specific forms {(value, from, to)} and {(value, {(from, to)})}

For example, expand operation on a set of the first form will expand a time interval into many time point intervals, each of which is an interval whose starting time point is the same as the ending time point. On the contrary, *coalesce* operation on a set of the second form will coalesce two intervals if they are adjacent or overlapping.

5) basic relational operations extended:

⑮ temporal versions of the basic relational operations such as join and selection

In summary, among these operations to be defined for the convenience of users, some of them can be defined as methods of some class in time library, and some others are defined as methods of a special abstract class, which is a superclass of all the classes for the time-varying attributes. The time library can be depicted as Figure 2, in which numbers represent the operations just introduced in this section. Operations corresponding to the numbers (7 and 12) missing in the Figure (e.g., time slice, rollback) and temporal versions of basic relational operators are supported by the new SELECT statement as special clauses. TOPs (stands for temporal operators) represents the abstract superclass of all the classes for the time-varying attributes.

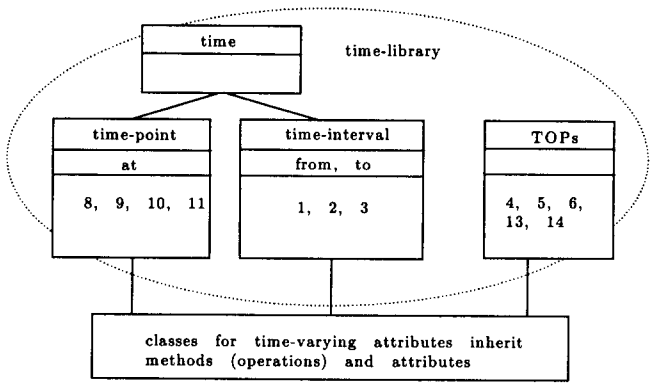

Figure 2: a time library and TVA classes

4.2 Syntax of a New SQL Select Statement

A new SELECT statement to be used to retrieve information from a temporal database has the following syntax. Its semantics is similar to that of the snapshot version, except for a few new clauses which are added to support the manipulation of time-varying information.

```
select          target_list
from            class_list
where           conditional_expression
moving window   window_time-span
when            temporal_condition
valid during    valid_time_period
as of           transaction_time_point
group by        attribute_list
having          conditional_expression_with_aggregate_function
```

Moving window clause is added to make it easy to do trend analysis during a given time interval, by computing and comparing values for each sub-interval of the interval. *When* clause is used to specify a temporal condition, which is a condition expressed in terms of time-varying attribute. This corresponds to the temporal version of the *restriction* operation in the relational data model. This *when* clause and the *from* clause having two or more classes can be used to perform a temporal join operation. *Valid during* clause is used either to specify the time interval for the time-slicing operation or to assign the valid time interval for the retrieved information. Finally, *as of* clause is used when one wants to carry out the rollback operation to a past time point.

4.3 Temporal Queries

The above explanation of our temporal query language can be complemented by issuing temporal queries against some temporal database. The schema and the database that we have defined in section 3 are to be referenced in this section.

query 1: "What was Kim's salary in Oct., 1992?"

```
select      st.salary
from        (select salary-t from employee
```

```
                    where name = 'Kim') as t(st)[1)]
when                getTime(st) overlap (10/01/1992, 10/31/1992);
```

In the above, 'from' clause has a subquery, whose result is defined as a table, t(st), which is a table, t, having a single column, st. t(st) is a table of OID's, each of which is an ID of an instance, belonging to the class salary-t. The above query first calculates t(st), and then for each st (e.g., OID) in the table, it checks the condition specified in the 'when' clause, to see whether the corresponding instance has a valid-time interval overlapping with the interval (10/01/1992, 10/31/1992). The valid-time interval is calculated by invoking the getTime method, inherited from TOPS class (see Figure 2). This query returns the salary of the instance which satisfies the condition. If one wants to know the same information as is known in January 1993, the next query will do.

query 2: "What was Kim's salary in Oct., 1992, as best known in Jan. 1993?"

```
select              st.salary
from                (select salary-t
                    from employee
                    where name = 'Kim') as t(st)
when                getTime(st) overlap (10/01/1992, 10/31/1992)
as of               1/1993;
```

The 'as of' clause has the effect of ignoring all the transactions that were performed after January, 1993. This query will be processed in the same way as query 1 is processed, except that the last three instances of the employee class will not be taken into account in this case.

query 3: "When has Kim moved to D2 department?"

```
select              getFrom(dt)          /* returns the starting point of an interval */
from                (select department-t
                    from employee
                    where name = 'Kim') as t(dt)
where               dt.department = 'D2';
```

1) This feature of defining the result of a subquery as a table is supported by UniSQL/X.

query 4: "Retrieve all the information of Kim during 1992 and 1993."

```
select          *
from            employee
where           name = 'Kim'
valid during    (1/1992, 12/1993);
```

'valid during' clause is used to slice the database along the time dimension. This is so-called time-slice operation. If one wants to get information about Kim for the last five years, the valid during clause should be changed to 'valid during (now - 5years, now)', where 'now - 5years' is the day 5 years ago from now, which can be calculated by invoking a method defined in the time-point class of the time library.

query 5: "How much is Kim's average salary when he was in D1 department?"

```
select          Avg(st.salary)
from            (select salary-t
                 from employee
                 where name = 'Kim') as t(st),
                (select department-t
                 from employee
                 where name = 'Kim') as t(dt)
where           dt.department = 'D1'
when            getTime(st) overlap getTime(dt);
```

This query first computes t(st) and t(dt) by selecting the salary-t and department-t attributes of the employee whose name is Kim, retrieves st's which satisfy the conditions specified in the 'where' and 'when' clauses, then returns the average salary of those st's.

query 6: "How long had Kim been in D1 department?"

```
select          getNoMonths(getTime(dt))
from            (select department-t
                 from employee
                 where name = 'Kim') as t(dt)
```

```
where          dt.department = 'D1';
```

getNoMonths returns the number of months during a time interval, which is returned by the *getTime* operation.

query 7: "When has Kim's salary been raised the most?"

```
select          max(st.salary - next(st).salary), window
from            (select sortTime(expandYear(salary-t))
                 from employee
                 where name = 'Kim') as t(st)
moving window   1 year;
```

expandYear operation expands the time interval of salary-t instance and *sortTime* operation sorts its result by the time. This kind of query could be useful, especially when performing a trend analysis during a certain period of time interval.

Other calculations that are too complex to be dealt with by a single statement, can be performed, using a nested query, set operations, interpreter variables to store temporary result, or their combination, as is used in the conventional systems.

5. Related Works

Though most researches on temporal database are done on the basis of the relational data model, there are some which take the object-oriented data model as the underlying data model. An object-oriented temporal model [16] is one based on object-based ER model. They have defined a type lattice, whose root is Object, under which there are Primitive-types (e.g., integer, real, etc), Collections (e.g., set[T], tuple, sequence[T]), and Class. New types are added as follows: Time is added as a subclass of Primitive-types, TimeSequence[T] as a subclass of Sequence[T], and NV-class and V-class as subclasses of Class. Classes of time-varying objects are defined as subclasses of V-class, and domains of their time-varying attributes are defined as subclasses of TS[T]. Attribute timestamp is attached to the instances of those subclasses, and three different kinds of time are stored as a timestamp: valid time interval, record time instant, and event time instant. SQL is extended into a temporal object-oriented SQL, TOSQL, which has new temporal clauses such as WHEN, TIME-SLICE, and MOVING-WINDOW. Messages can appear in SELECT and WHERE clauses, and

messages can be nested. FROM clause of the standard SQL is replaced by the FOR EACH clause, in order to refer to a particular object. Notice that attaching three different kinds of timestamp (transaction time, valid time and event time) to an attribute makes it difficult to perform rollback operations and that the event time may not be so useful except for a specific application. Also, notice that this model does not allow the same timestamps to be shared by the synchronous attributes.

In [22], properties of time-varying objects are modeled as functions, each of which returns another function, that returns, given a time object, the value of the corresponding property. They have defined a hierarchy of abstract time types, with *point* as a supertype of all types, and another hierarchy of set types, which are defined for each time type, with *{point}* as supertype of them. Their model supports both attribute timestamp and object timestamp. Because no special constructs (e.g., those for slice, rollback, etc) are defined for temporal queries, the same language can be used for both temporal and non-temporal queries. However, query optimization may become more difficult, and their queries, which are basically nested-for loops, may be uncomfortable to the users of relational or object-oriented database systems, who have been using SQL-type query languages.

Another temporal model is defined based on the concept of time sequence in [21]. Time sequence is a sequence linearly ordered in time, and a collection of time sequences for the objects that belong to the same class is called time sequence collection, TSC. Each TSC has three components, surrogate domain, time domain and attribute domain and is defined to have some properties such as granularity, lifespan, and interpolation function type. A data manipulation language is defined which is a variant of SQL but provides diverse commands for data manipulation. Though their model is independent of any logical model, they have shown that if their model is represented in the relational model, they cannot avoid performing join operations to get information of an object when it has several temporal attributes. As such, they have defined another concept, family of TSCs with same surrogate. Instead of representing their model in relational model, it would be natural to represent it in object-oriented model and then there would be no need to define the concept, family.

6. Summary and Future Works

Having discussed the most essential issues that are common to many research papers on temporal database, this paper illustrated why the relational data model is not appropriate as a data model to support the manipulation of time-varying information. Then, a way of extending the object-oriented data model is explained using a sample database which is assumed to be created from a set of sample transactions. In the extended data model, attribute timestamp is used for valid time interval, while instance timestamp is used for transaction time point. Users are allowed to specify whether a class is to be either historical or temporal. The extended data model is further explained by issuing temporal

queries against the sample database, following the syntax of the new select statement. To provide convenience to the users of temporal databases, operations that seem to be used frequently by the users are collected into a time library.

Although we leave out the detail explanation on how relational operations can be integrated in the temporal queries and on the semantics of newly added temporal operations, it may be desirable to add such explanation, to some extent, with examples.

Compared with related works, our work has the following characteristics: 1) it has the flexibility of allowing syhchronous time-varying attributes to be defined so that they share the same timestamp, which makes querying in terms of such attributes simpler than otherwise, 2) it is at user's disposal to define a class either to be historical or to be temporal, and 3) typical operations required for the manipulation of time-varing information are collected into a time library, though they need more collection and tuning thereof.

Many things remain to be done. Modal operators such as *always*, *often* and *since*, need to be defined so that they can be used in a variety of query expressions. Also, there should be further research on temporal constraints, temporal query optimization, storage structure and access methods for the temporal database, and so on. Researches on temporal database and on multidatabase could be utilized for the creation and manipulation of data warehouse, which is built to be used for corporate decision making.

REFERENCES

[1] M. Atkinson, F. Bancilhon, D. Dewitt, K. Dittrich, D. Maier, and S. Zdonik, *The Object-Oriented Database System Manifesto*, Elsevier Science Publishers, Amsterdam, 1990

[2] J. Ben-Zvi, *The Time Relational Model*, Ph. D.. thesis, Computer Science Department, UCLA, 1982

[3] J. Clifford and A. Croker, "The Historical Relational Data Model (HRDM) and Algebra based on Lifespan," *proceedings of the third International Conference on Data Engineering*, pp Feb., 1987

[4] H.-T. Chou and W. Kim, "A Unifying Framework for Version Control in a CAD Environment," *VLDB*, 1986

[5] F. Clifford and A. U. Tansel, "On an Algebra for Historical Relational Databases: Two Views.", *proceedings of ACM SIGMOD*, pp 247-265, May 1985

[6] J. Clifford and D. S. Warren, "Formal Semantics for Time in Databases," *ACM TODS* 8(2), June 1983

[7] Shashi K. Gadia and Sunil S. Nair, "Temporal Databases: A Prelude to Parametric Data.", in *Temporal Databases* ed. by Tansel et al., 1993

[8] S. K. Gadia, "A Homogeneous Relational Model and Query Languages for Temporal Databases," *ACM TODS*, 13(4), Dec. 1988, pp 418-448

[9] S. K. Gadia and C. S. Yeung, "A Generalized Model for a Relational Temporal Databases," *ACM SIGMOD*, 1988

[10] Christian S. Jensen and Richard T. Snodgrass, "Semantics of Time-Varying Information", *Information Systems*, 21(4), pp 311-352, 1996

[11] R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases," *ACM Computing Surveys*, 22(4), pp 376-408, Dec. 1990

[12] Nikos A. Lorenztos, "The Interval-extended Relational Model and Its Application to Valid-time Databases," in *Temporal Databases* ed. by Tansel et al., 1993

[13] E. McKensie Jr. and R. Snodgrass, "Evaluation of Relational Algebras Incorporating the Time Dimension in Databases," *ACM Computing Surveys*, 23(4), pp 501-543, Dec. 1991

[14] S. B. Navathe and R. Ahmed, "A Temporal Relational Model and a Query Language," *Information Sciences*, Vol. 49, pp 147-175, 1989

[15] G. Ozsoyoglu and R. T. Snodgrass, "Temporal and Real-time Databases: a survey", *IEEE Transactions on Knowledge and Engineering*, 7(4), pp 513-532, 1995

[16] Ellen Rose and Arie Segev, "TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints", 10-th International Conference on ER Approach, Oct., 1991

[17] R. Snodgrass and I. Ahn, "Temporal Databases," *IEEE Computer*, 19(9), pp 35-42, Sep. 1986

[18] N. L. Sarda, "HSQL: A Historical Query Language", in *Temporal Databases* ed. by Tansel et al., 1993

[19] E. Sciore, "Using Annotations to Support Multiple Kinds of Versioning in an Object-Oriented Database system," *ACM TODS*, 16(3), pp 417-438, 1991

[20] E. Sciore, "Versioning and Configuration Management in an Object-Oriented Data Model," *VLDB Journal*, 3(1), pp 7-106, 1994

[21] Arie Segev and Arie Shoshani, "A Temporal Data Model Based on Time Sequences", in *Temporal Databases* ed. by Tansel et al., 1993

[22] G. Wuu and U. Dayal, "A Uniform Model for Temporal Object-Oriented Databases," *Proceedings of the eighth International Conf. on Data Engineering*, pp 583-593, 1992