SPATIO-TEMPORAL DATABASE SUPPORT FOR LEGACY APPLICATIONS

Michael Böhlen[†] Christian S. Jensen[†] Bjørn Skjellaug[‡]

[†] Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E DK-9220 Aalborg Øst, Denmark
{boehlen,csj}@cs.auc.dk

[‡] SINTEF Telecom and Informatics, Oslo
and
Department of Informatics, University of Oslo
Gaustadalléen 23, N–0371 Oslo, Norway
bjornsk@ifi.uio.no

KEYWORDS: temporal, spatial, and spatio-temporal data; SQL; legacy software; multi-dimensional data; language design.

ABSTRACT

In areas such as finance, marketing, and property and resource management, many database applications manage spatio-temporal data. These applications typically run on top of a relational DBMS and manage spatio-temporal data either using the DBMS, which provides little support, or employ the services of a proprietary system that co-exists with the DBMS, but is separate from and not integrated with the DBMS. This wealth of applications may benefit substantially from built-in, integrated spatio-temporal DBMS support. Providing a foundation for such support is an important and substantial challenge.

This paper initially defines technical requirements to a spatio-temporal DBMS aimed at protecting business investments in the existing legacy applications and at reusing personnel expertise. These requirements provide a foundation for making it economically feasible to migrate legacy applications to a spatio-temporal DBMS. The paper next presents the design of the core of a spatio-temporal, multi-dimensional extension to SQL–92, called STSQL, that satisfies the requirements. STSQL does so by supporting so-called *upward compatible*, *dimensional upward compatible*, *reducible*, and *non-reducible* queries. In particular, dimensional upward compatibility and reducibility were designed to address migration concerns and complement proposals based on abstract data types.

1 INTRODUCTION

A wide range of applications manage spatial, time-varying, or spatio-temporal data. Typically, CAD and GIS applications maintain huge volumes of spatio-temporal data, i.e., data that includes spatial extents, shapes, or locations of objects, and time-related versioning of data. Financial and record-keeping applications such as accounting, banking, personnel management, and medical records, manage large amounts of time-varying data.

A common characteristic of applications such as these is that the semantics of spatial and time-varying data are the responsibility of and are encoded solely in the applications or some proprietary system [13, 20, 25]. That is, the semantics of the spatial and temporal dimensions, which are intrinsic properties of the data, are unknown to the underlying DBMS. Thus, spatio-temporal applications do not currently enjoy the built-in, integrated support that current DBMS's supply to less challenging applications. This paper addresses the challenge of providing spatio-temporal DBMS support to spatio-temporal data management applications.

The database technology in the commercial market is not yet close to incorporating the necessary spatio-temporal capabilities. However, over the past decade or two, substantial research efforts in the areas of temporal and spatial data management have resulted in a substantial number of proposals for temporal and spatial data models and query languages (e.g., IXSQL [14], TempSQL [9], TSQL2 [24], ROSE Algebra [10], ParaSQL [5], Spatial SQL [7], and GEO System [18]). But, none of these proposals address the migration of legacy applications to a spatio-temporal DBMS.

The paper defines a requirement aimed at guaranteeing that legacy DBMS application code, with its associated data, without changes remains operational when migrated to the spatio-temporal DBMS. Another requirement aims at ensuring that new application code that exploits the new spatio-temporal support of the DBMS may co-exist harmoniously with the legacy code. Finally, a requirement aims at ensuring that programmers familiar with SQL–92 may start using the new features of the DBMS without a need for expensive training.

This paper was presented at SAC'98, the 1998 ACM Symposium on Applied Computing, http://www.acm.org/conferences/sac/sac98/

STSQL supports the two generic temporal aspects, *valid time* and *transaction time*, of database facts that record when facts are true in the modeled reality and when they are current in the database, respectively. STSQL also supports the management of arbitrary spatial aspects of information (we provide one generic *space* aspect because the distinction between valid and transaction time does not apply well to space). The time and space values are recorded by *timestamps* and *spacestamps* that are associated with tuples as values of special attributes, and multiple space and time dimensions are permitted in a single table.

The migration requirements that dictate the general properties of STSQL were originally developed in the context of bitemporal tables, i.e., tables supporting, at most, one transaction time and one valid time [2, 3]. STSQL supports multiple valid- and transaction-time and multiple space attributes in a single dimensional table. We are aware of no other models with this property. Among the few spatio-temporal data models that exist, ParaSQL [5] may be the closest relative of STSQL. However, being based on an attribute-value stamped data model, ParaSQL differs substantially from STSQL; apart from upward compatibility, it does not satisfy any of the migration requirements. STSQL generalizes ATSQL [3] and proposed additions to the SQL/Temporal part of the SQL3 standard [22, 23], which support bitemporal tables and satisfy temporal migration requirements. Considering spatial data models, we have found no data models that provide migration support beyond upward compatibility. The SQL-based languages GEOQL [17], PSQL [19], KGIS [11] and Spatial SQL [7] preserve the non-dimensional SQL and satisfy upward compatibility, and they define explicit extensions to the SQL select statement for the handling of spatial values. KGIS and Spatial SQL also define, outside SQL, other language constructs to augment the spatial capabilities of their models and languages.

The paper is organized as follows. Following an introduction in Section 2 of a case that will be used for illustration throughout, Section 3 defines three fundamental requirements to a spatio-temporal data model and query language. Section 4 proceeds by presenting the design of the spatio-temporal extension STSQL of SQL–92 that satisfies the requirements. Section 5 concludes the paper and outlines open research issues.

2 A SPATIO-TEMPORAL DATA MANAGEMENT APPLICATION

The case example presented here is based on an existing legacy planning and scheduling system (termed Ecoplan) used for forest management, specifically for long-term forest harvest scheduling based on ecological, recreational, and economical constraints [16].

While the system has four modules, we focus on the data module, which at present manages data in a loosely coupled fashion. Spatial data is stored in files and is managed by the module using proprietary data structures. The associated textual and numeric property data is managed by a relational DBMS.

Using examples from this case, we will exemplify the design of a spatio-temporal relational data model step-by-step. To concisely illustrate the contributions of this paper, we have substantially simplified the system. We will thus assume that the system's database contains three tables as shown in Figure 1.

`stands:`

st_ID	index	specie	planted
st_100	high	pine	1935
st_230	high	birch	1957
st_245	low	birch	1946
st_560	high	spruce	1963

`plans:`

pl_ID	st_ID	volume	ripe
pl_29	st_100	2000	2000
pl_29	st_560	900	2000
pl_29	st_230	1500	2002
pl_34	st_245	400	2010

`estates:`

es_ID	owner
es_34	Paul
es_401	Mary
es_63	Mary
es_80	Peter

Figure 1: A Case Example Database

The `stands` table to the left captures data about regions that are homogeneous with respect to soil fertility (a so-called index), wood specie, and average age (recorded as the year the trees were planted). Thus, a tuple in `stands` records surveyed data about a forest region; the `estates` table to the right records the IDs of estates and their owners. An estate is a legal entity covering a geographical region, possibly including one or more forests. Finally, the `plans` table in the middle defines the harvest plans for stands, with each stand being associated with one or more plans (and vise versa), an estimated harvest volume in m^3 for each stand, and an optimal harvest time (a so-called ripe year) of the stand. Thus, a plan of a stand is a calculation based on the stands data and specific scheduling parameters. Figure 2 illustrates the spatial locations of estates and stands, and it also indicates the plans of stands. (Regions st_100A, st_100B, and st_100C are subregions of st_100 that will be computed by subsequent examples.)

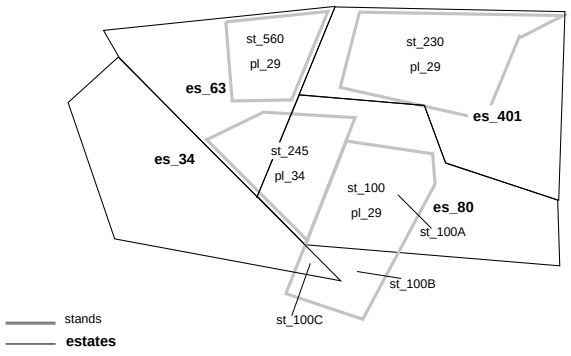

Figure 2: Distribution of Estates and Stands with Related Plans

3 MIGRATION REQUIREMENTS

This section defines and discusses three important requirements to a spatio-temporal data model and query language. While space and time are quite different aspects of data, the requirements are able to treat the two aspects uniformly.

3.1 Overview

When migrating to a new DBMS, it is desirable, or even essential, to protect existing investments in legacy application code and in programmer expertise. Informally stated, it is important that

1. all non-spatio-temporal legacy data is maintained by the new DBMS;
2. all non-spatio-temporal legacy code (i.e., queries and modifications) remains operational using the new DBMS, and it may access the same data as before;
3. skilled legacy system developers should with little effort be able to utilize a core subset of the added functionality in the spatio-temporal DBMS; and
4. the spatio-temporal DBMS should provide constructs to utilize the full potential of a spatio-temporal data model and query language.

These requirements should be supported in concert by the data model and query language of a spatio-temporal DBMS (STDBMS). The next step is to make the requirements precise. To do so, some notation is needed.

We will assume that a data model, M, is given by a query language component, QL, and a component of data structures, DS, manipulated by the query language. A data model captures the functionality of a DBMS that implements the data model. In the relational model, the most important user-level query language is SQL, and the table is the central data structure. Next, for a query language expression s and an associated database *db*, both legal elements of data model $M = (DS, QL)$, we define $\langle\langle s(db)\rangle\rangle_M$ as the result of applying s to *db* in data model M. We use the superscripts "^{s}" and "^{d}" to indicate snapshot and dimensional entities, respectively. For example, q^d denotes a dimensional query. The dimensional slice operator, $\tau_p^{M^d,M^s}$, where p, a dimensional point, is a parameter, takes a dimensional database db^d (in the data model M^d) as argument. It returns a snapshot database db^s (in the data model M^s) containing all tuples that are defined at point p. In other words, db^s consists of snapshot tables of the tuples (of tables) in db^d, but without their dimensional attributes, whose associated (dimensional) regions as defined by the combination of their dimensional attributes include the point p.

3.2 Compatibility Requirements

The first two informal requirements above address upward compatibility issues, and are formally defined in the following. We first define *upward compatibility*.

Definition 3.1 (UC) Model M_1 is *upward compatible* with model M_2 iff

- $\forall db_2 \in DS_2\ (db_2 \in DS_1)$,
- $\forall s_2 \in QL_2\ (s_2 \in QL_1)$, and
- $\forall db_2 \in DS_2\ (\forall s_2 \in QL_2\ ($
  $\langle\langle s_2(db_2)\rangle\rangle_{M_2} = \langle\langle s_2(db_2)\rangle\rangle_{M_1}))$. ∎

Upward compatibility captures the conditions that need to be satisfied in order to allow a smooth transition from a current system, with data model M_2, to a new system, with data model M_1. The first two conditions imply that all existing databases and query expressions in the old system are also legal in the new DBMS. The last condition guarantees that all existing queries compute the same results in the new and the old DBMS.

The second compatibility requirement, *dimensional upward compatibility*, ensures that legacy applications remain operational even if the database is rendered dimensional. Intuitively, the requirement is that a query q^s must return the same result on an associated snapshot database db^s as on the dimensional counterpart of the database, $\mathcal{D}(db^s)$ (with operator $\mathcal{D}$ adding dimensions to its argument database). Moreover, modifications should not affect this. We use $\mathcal{U}$ to denote a sequence of modifications.

Definition 3.2 (DUC) Model M^d is *dimensional upward compatible* with model M^s iff

- M^d is upward compatible with M^s and
- $\forall db^s \in DS^s\ (\forall\, \mathcal{U}\ (\forall q^s \in QL^s\ ($
  $\langle\langle q^s(\mathcal{U}(db^s))\rangle\rangle_{M^s} = \langle\langle q^s(\mathcal{U}(\mathcal{D}(db^s)))\rangle\rangle_{M^d})))$. ∎

To satisfy this requirement, all dimensional attributes must be managed specially in legacy queries and modifications (remember that legacy applications accessing the database are not aware of them).

For time dimensions, the implicit handling of timestamps has been investigated carefully [2]. Legacy query expressions are evaluated only on tuples with valid and transaction times that overlap with *now*. Legacy modification statements are slightly more complicated. Such statements affect current and future data only. Thus, newly inserted tuples get valid and transaction time periods from the (constant) time of the insertion until the (variable) current time, whereas logical deletions set the end times of periods of tuples to be deleted to the time of the deletion, thus removing the tuples from all future current states.

In contrast to the temporal dimensions, there are no obvious default values for spatial dimensions. We have decided to let legacy queries ignore spatial dimensions. This is consistent with how spatial dimensions are handled when spatial values are captured using explicit attributes: if such attributes are not mentioned explicitly, they are ignored. Legacy modifications set space values of tuples to some default value, i.e., either a user-defined default or the system default.

To illustrate the compatibility requirements, consider the following three statements issued in an STDBMS that satisfies UC and DUC.

```
> SELECT * FROM plans;
> ALTER TABLE plans
              ADD harvest1 PERIOD AS VALID;
> SELECT * FROM plans;
```

The first statement is an SQL–92 query issued on the legacy table, `plans`. Due to UC, it returns the same result as it did in the old DBMS. The next statement exemplifies operator $\mathcal{D}$ from above. It alters the `plans` table by adding a valid-time dimension to indicate harvest periods of stands, perhaps because a new application needs this information about plans. The last statement is now on an extended table, but due to DUC it yields the same result as the first statement. In particular, it does not return the harvest period dimension so that legacy applications do not have to be changed.

Summarizing, a UC evaluation is simply an evaluation that is identical to that of the legacy DBMS, and a DUC evaluation simulates a non-dimensional database where only one state is maintained.

3.3 Reducibility Requirements

To naturally generalize the snapshot relational model to a dimensional relational model, we adopt the view that a dimensional table is a collection of snapshot tables, with each snapshot table having an associated multi-dimensional point and containing all the snapshot tuples that have an associated multi-dimensional region that contains the point.

We first define what it means for a data model to be *snapshot reducible* with another data model.

Definition 3.3 (SR) Data model M^d is *snapshot reducible with respect to* data model M^s iff

$$\forall q^s \in QL^s\ (\exists q^d \in QL^d\ (\forall db^d \in DS^d\ (\forall p\ (\tau_p^{M^d,M^s}(q^d(db^d)) = q^s(\tau_p^{M^d,M^s}(db^d)))))).$$ ∎

This concept of snapshot reducibility generalizes the similar concept from temporal databases in a straight-forward manner [21]. Observe that q^d being snapshot reducible with respect to q^s poses no syntactical restrictions on q^d. It is thus possible for q^d to be quite different from q^s, and q^d might be very involved. This is undesirable, as we would like the dimensional model to be a straight-forward extension of the snapshot model. Consequently, we require that q^d be a *syntactically similar snapshot reducible* extension of q^s [4].

Definition 3.4 (SSSR) Data model M^d is a *syntactically similar snapshot-reducible extension* of model M^s iff

- data model M^d is snapshot reducible with respect to data model M^s, and
- there exist two (possibly empty) strings, S_1 and S_2, such that each query q^d in QL^d that is snapshot reducible with respect to a query q^s in QL^s is syntactically identical to $S_1 q^s S_2$.

If the two strings S_1 and S_2 are both the empty string, the extension is termed a syntactically identical snapshot reducible extension. ∎

This requirement makes it possible for the SQL–92 programmer to easily formulate spatio-temporal queries. To illustrate this, we first extend the `estates` and `stands` tables with two-dimensional valid-space attributes and then issue three spatio-temporal queries, which are explained next.

```
> ALTER TABLE estates
              ADD es_area 2D_REGION AS SPACE;
> ALTER TABLE stands
              ADD st_area 2D_REGION AS SPACE;

> REDUCIBLE (es_area) AS area
    SELECT * FROM estates;
> REDUCIBLE (es_area, st_area) AS area
    SELECT es_ID, st_ID
    FROM   estates, stands;
> REDUCIBLE (es_area, st_area) AS area
    SELECT st_ID
    FROM stands
    WHERE NOT EXISTS (
      SELECT *
      FROM estates);
```

Note that the queries have an SQL–92 core and are prepended with a `REDUCIBLE` string. The string, termed a *flag*, indicates how to handle the dimension attributes in the queries. Flags in STSQL is an important topic of the next section.

The presence of the `REDUCIBLE` flag implies that, conceptually, all queries are computed in a point-by-point fashion. More specifically, for each point in space, the legacy SQL statement following the flag is evaluated on the snapshot database corresponding to that point. Next, the results for each point in space are integrated into a single dimensional table: Tuples with identical explicit attributes are replaced by a single tuple with the same explicit attributes values and an attribute `area`, which stores the region corresponding to the union of all the tuples' associated points in space. (Here, we assume that the data type used for spatial regions, i.e., the data type of `area` is capable of representing any union of points in space. If this is not the case, several tuples are generally needed for capturing the spatial region.)

We assume that all spatial values for estates and stands, shown in Figure 2, have been included into the database in Figure 1. Then the first statement returns all tuples of `estates`. With each tuple an `area`-attribute that specifies the estate's region will be returned. The second query retrieves for each point in space the respective estate and stand. More precisely, the result of the query contains the tuples $\langle es\_34, st\_245, reg_{es\_34} \cap reg_{st\_245}\rangle$ and $\langle es\_80, st\_245, reg_{es\_80} \cap reg_{st\_245}\rangle$. The third statement is conceptually not different from the other ones. At each point in space it retrieves stand IDs if there does not exists an estate. In other words, the query determines those (parts of) stands that are not located within an estate. The result is $\langle st\_100, reg_{st\_100B}\rangle$.

In summary, a snapshot reducible query generalizes a snapshot query by reducing argument dimensional tables to point-indexed snapshot tables, then computes the corresponding snapshot query on those snapshot tables, and finally "unions" the snapshot results to achieve a dimensional result table. The main characteristic of snapshot-reducible evaluation is its point-based nature, where dimensional tables may be seen as indexed sequences of snapshot tables. Hence, the key word `REDUCIBLE`.

A spatio-temporal query language should also provide queries

that have no counterparts in the snapshot query language. That topic is considered next.

3.4 Beyond Reducibility

Reducible STSQL queries perform computations on the dimension attributes as specified by reducibility and by the SQL–92 queries they reduce to. The advantage is that it is easy to immediately write a wide range of dimensional queries that perform potentially complex manipulation of dimension attributes. But many reasonable and useful dimensional queries cannot be specified as reducible generalizations of snapshot queries, so there is a need for the ability to specify queries where no processing of the dimension attributes is hard-wired into the data model, but where the programmer instead has complete control over the manipulation of the dimension attributes.

We thus make it possible to specify in the flag of a statement that dimension attributes should simply be considered as regular attributes. In addition, we provide a range of predicates and functions that operate on the data types of the dimension attributes. This gives the programmer full control over the dimension attributes. For example, non-reducible queries may relate database states that apply to different points in multi-dimensional space. For this reason, we use the key word `NONREDUCIBLE` to indicate dimension attributes that should be treated as regular attributes in a query. An example follows.

```
> NONREDUCIBLE (es_area, st_area)
    SELECT s.st_ID, s.st_area
    FROM   stands s
    WHERE  NOT EXISTS (
      SELECT *
      FROM estates e
      WHERE e.es_area CONTAINS s.st_area);
```

The query retrieves all stands for which no single estate exists that covers the stands area. In this query, we consider the regions of the stands as being non-decomposable and constrain them with a spatial predicate. This contrasts the `REDUCIBLE` queries from before, where regions are decomposed into their constituent points.

3.5 Summary

We have introduced three ways of handling dimension attributes in spatio-temporal tables. Dimension attributes that are not mentioned in the flag of a query language statement are "ignored," or treated consistently with dimension upward compatibility. If the key word `REDUCIBLE` is used for dimension attributes, they are treated as implicit dimensions of data, and the statement is evaluated with semantics that meet the snapshot reducibility requirement. This provides built-in spatio-temporal query processing. Finally, if the key word `NONREDUCIBLE` is used for dimension attributes in the flag of a statement, the dimension attributes are treated as regular attributes. This provides maximum flexibility in writing spatio-temporal queries.

4 STSQL DESIGN

This section discusses the design of a spatio-temporal extension to SQL–92 based on the requirements presented in Section 3. We briefly discuss the new data types of STSQL, then explore in more detail its syntax and semantics.

4.1 Space and Time Data Types

The initial step in the design of STSQL is to introduce new data types that capture time and space values. For time values STSQL uses anchored time periods. Spatial values are unions of regions. Regions are either defined over 1-, 2-, or 3-dimensional spatial domains. The corresponding data types are `PERIOD, 1D_REGION, 2D_REGION` and `3D_REGION`, respectively. (In this paper, the number of different region data types and their individual characteristics are of minor importance. The interested reader is referred to, e.g., Güting [10] for more details about a variety of spatial data types).

The new data types must be accompanied by predicates and functions that operate on them. Again, the specific choice and number of these is not important for the contribution of this paper, so we simply give a list of names and brief informal descriptions of some useful, representative predicates and functions, see Figure 3.

The predicates for periods and regions should be well known to those familiar with, e.g., Allen's interval logic [1] and Egenhofer and Franzosa's point-set topological spatial relations [8].

4.2 Dimensional Tables and Databases

The next step is to make tables *dimensional*, in order to provide a basis for built-in dimensional support for modifications and queries in the query language. The data types introduced in the previous section are utilized. Note that the data types, like any other SQL–92 data types, may be employed for defining domains of attributes that are no different from regular attributes. Including such attributes in a table does not render the table dimensional; rather, the table is a regular table that includes regular attributes, some or all of which happen to be of type `PERIOD`, `1D_REGION`, `2D_REGION`, or `3D_REGION`. The DBMS attaches no special semantics to these attributes.

To provide built-in dimensional support, e.g., dimensional upward compatibility and snapshot reducibility, it is necessary to be able to designate certain time or space valued attributes as special dimensional attributes. Tables with such attributes are then dimensional tables. In STSQL, dimensional tables may have any number of dimensional attributes, and each dimension attribute may be of any of the four new time and space types introduced in Section 4.1. In addition, a dimension attribute is specified as either a `VALID`, a `TRANSACTION`, or a `SPACE` attribute. We then obtain three conceptually different types of dimension attributes. With `d_att` being the name of a dimension attribute, the three types are as follows (where `x` denotes `1`, `2`, or `3`).

```
d_att PERIOD AS VALID
d_att PERIOD AS TRANSACTION
d_att xD_REGION AS SPACE
```

In typical use, a dimension value of a tuple is associated with the tuple as a whole. In the first type, `d_att` then records when some temporal aspect of the information recorded by the (non-dimensional) attribute values of the tuple as a whole is true,

name	description	domain	value
BEGIN/END	timestamp start/end time	period	time instant
MEETS	adjacency/neighbor	period/region	boolean
OVERLAPS	sharing common period/region	period/region	boolean
CONTAINS	one within the other	period/region	boolean
PRECEDES	one strict earlier than the other	period	boolean
INTERSECTION	shared period/region	period/region	period/region
DURATION	length of period in specified units	period	a number
AREA	number of square units	region	a number

Figure 3: Some predicates and functions

or valid, in the mini-world. For example, we have previously added a `harvest1` attribute to the `plans` table, recording the harvest period for a plan. While with the first type above, we record when some temporal aspect of a tuple is valid, the second type records when a tuple is current in the table, or, equivalently, when we believed in the information recorded by the tuple. This transaction-time aspect of a tuple is important in applications that require accountability or traceability of database modifications. In contrast to valid-time values, which are determined by the mini-world modeled by the database, the transaction-time values are determined by the modification activity on the database. Because the merits of the distinction between valid and transaction time are unclear for space, we provide support for a single, generic spatial aspect. As an example, we have previously added the attribute `es_area 2D_REGION AS SPACE` to the `estates` table. This dimensional attribute is intended to record the geographic areas of individual estates.

In contrast to most spatial and temporal models, STSQL permits multi-dimensional tables where a single table may have any number of dimension attributes of any of the types explored above. This added generality is useful for many purposes. Several valid-time attributes are useful, e.g., when the information of a tuple is true in several different (possible) worlds. For example, different historians, archeologists, or interest groups may possess different, competing world views, all of which could be represented in a single table. Several `VALID`-type attributes may also record different temporal aspects of a tuple. For example, the `plans` table previously presented had a `VALID` attribute `harvest1` recording when a stand is supposed to be harvested. We can also add a new `VALID` attribute denoting when the textual property data about a plan for a stand are valid. Certainly, these two attributes record different aspects of a plan. We may also add a `VALID` attribute recording an alternative harvest period that denotes a harvest period of a stand calculated using a different method and different parameters. The resulting two harvest attributes reflect different (possible) worlds. Considering space instead of time, it is equally easy to envision uses of multiple dimension attributes: The multiple-worlds argument applies equally well to space, and tuples may have several different kinds of spatial aspects. Couclelis discusses issues related to these [6]. Reasons for recording multiple transaction attributes have been explored elsewhere [12]. The choice of how to use multiple valid, transaction, and space attributes is up to each specific application.

In summary, we have added multiple space and time dimensions to tables, thereby obtaining the notation necessary to enable dimensional semantics to be built into modifications and queries. The next step is to explore the management of databases with multi-dimensional tables.

4.3 STSQL Statements

This section presents the core of STSQL. An EBNF syntax is given for the central extensions to SQL–92, and examples from the forest management application are used for illustrating the semantic properties of STSQL.

4.3.1 Alter and Create Statements

Legacy tables can be extended with spatial and temporal dimensions. For example, the following statements extend the `stands`-table from our case with further dimensions.

```
> ALTER TABLE stands ADD survey PERIOD;
> ALTER TABLE stands
            ADD st_vt  PERIOD AS VALID;
> ALTER TABLE stands
            ADD st_tt  PERIOD AS TRANSACTION;
```

These statements alter the `stands` table to include a valid-time dimension, a transaction-time dimension, and a user-defined attribute, the latter denoting the period during which a stand is surveyed. Note the difference between `survey` and `st_vt`. The former is not a dimension and, therefore, not subject to upward compatibility, dimensional upward compatibility, or reducibility.

4.3.2 Queries, Flags, and Dimension Identifiers

This section explores dimensional queries. All sample queries are evaluated on the tables shown in Figure 4. In order to understand the queries and modifications, it is essential to understand the semantics associated with these tables. We discuss each table in turn.

The `stands` table models the (surveyed and analyzed) status of stands. For each stand we record, e.g., the specie of the stand's dominant tree population, the soil fertility of the stand (i.e., the index), the stand's location, and a period of validity. A transaction time is used to retain a record of modifications. In stand st_100, pine trees have good growing conditions, i.e., high soil fertility. They were planted in 1935 and the stand was surveyed between 1984 and 1986. The stand location is the region reg_{st_100}. The information has been valid since 1989, but was first recorded in 1996.

The `estates` table records for each estate its owner, the validity period of the ownership, and the area that it covers. A transaction time is used to record modifications. During 1995 and

stands

st_ID	index	specie	planted	survey	st_vt	st_tt	st_area
st_100	high	pine	1935	1984-1986	1989-now	1996-now	reg_{st_100}
st_230	high	birch	1957	1984-1986	1989-now	1996-now	reg_{st_230}
st_245	low	birch	1946	1984-1986	1989-now	1996-now	reg_{st_245}
st_560	high	spruce	1963	1984-1986	1989-now	1996-now	reg_{st_560}

estates

es_ID	owner	es_area	es_vt	es_tt
es_34	Paul	reg_{es_34}	1995-now	1994-now
es_63	Mary	reg_{es_63}	1996-now	1996-now
es_80	Peter	reg_{es_80}	1996-now	1995-1996
es_401	Mary	reg_{es_401}	1996-now	1995-1996
es_80	Peter	reg_{es_80}	1996-1999	1997-now
es_401	Mary	reg_{es_401}	1996-1999	1997-now
es_100	Tom	$reg_{es\_80} \cup reg_{es\_401}$	2000-now	1997-now

plans

pl_ID	st_ID	volume	ripe	pl_vt	harvest1	harvest2
pl_29	st_100	2000	2000	1996-now	1998-2000	1999-2004
pl_29	st_560	900	2000	1996-now	1999-2001	2001-2003
pl_29	st_230	1500	2002	1996-now	2000-2002	2005-2008
pl_34	st_245	400	2010	1995-1996	2009-2011	2009-2011
pl_35	st_245	500	2011	1997-now	2010-2012	2010-2012

Figure 4: The Spatio-Temporal Example Database

1996, it was recorded that estate es_80, covering area reg_{es_80}, was owned by Peter from 1996 onwards. Similarly, it was recorded that estate es_401, covering area reg_{es_401}, is owned by Mary from 1996 onwards. In 1997, Mary and Peter agreed to sell their estates es_401 and es_80, respectively, to Tom, effective as of year 2000. Tom's estate will then cover the areas of these two estates.

The `plans` table records how stands are cultivated. For each stand, we record the volume to be harvested and the ripe year. Each plan has two harvest periods, calculated according to different scheduling methods that emphasize some growth conditions differently, e.g., according to soil fertility, climate, etc. Plan pl_34 schedules stand st_245 to be harvested from 2009 to 2011. The expected harvest volume is $400m^3$, and the ripe year is 2010. At some point, plan pl_34 for stand st_245 is superseded by plan pl_35. The new plan postpones the harvest period to 2010–2012 because, due to new climate estimates, the new expected ripe year has moved to 2011. The new expected harvest volume is 500 m^3.

The syntactic extensions to SQL–92 that are needed to formulate spatio-temporal statements are relatively few. The *flag* is the central novel construct and is used to indicate the desired evaluation mode(s) (cf. Section 3). Flags are placed in front of SQL statements and indicate whether the statements have to be evaluated according to reducible and/or non-reducible semantics. Additionally, it is possible to express domain restrictions and range specifications [3].

The following EBNF defines the syntax of flag. The `<cursor specification>` is the standard's production for the `SELECT` statement [15].

```
<cursor specification>
             ::= flags <query expressions>
flags        ::= flag { "AND" flag }
flag         ::= modifier dimensions
modifier     ::= "REDUCIBLE" | "NONREDUCIBLE"
dimensions   ::= "(" column_reference
                   { column_reference } ")"
                   [ "AS" <identifier> ]
```

The dimension(s) that a particular flag modifier applies to is (are) given by the non-terminal `dimensions` and have to follow the reducible or nonreducible modifier. Because of the multi-dimensional nature of STSQL, dimensions have to be named explicitly—unlike in frameworks with a fixed number of dimensions, this information cannot be inferred automatically.

To be meaningful, a reducible evaluation must apply to precisely one dimension from each argument table in the SQL statement. This requirement reflects the fact that a flag (and thus a reducible evaluation) applies to an *entire* statement. In general, no meaningful semantics can be given to reducible statements with tables that do not participate in the reducible evaluation. Note that derived table expressions (i.e., table expressions in the from clause) start a new scope whereas subqueries in the where clause do not. It should also be clear that the dimension types that take part in a reducible evaluation must be homogeneous. Reducible semantics are not meaningful when combining valid time and valid space or transaction time and valid time because of the different semantics associated with the respective dimensions.

When formulating queries on dimensional tables, it is advantageous to proceed in several steps. Initially, all dimensions are ignored and the core STSQL query, typically an SQL–92

query, is formulated. The next steps concern the formulation of the query's flag. For each dimension of each table in the query, we must determine and express in the flag the dimension's use in the query. First, we determine what dimensions should be evaluated with reducible semantics. Each occurrence of the `REDUCIBLE` keyword requires the participation of exactly one dimension from each table. Second, we determine which dimensions are to be given `NONREDUCIBLE` semantics. This semantics is chosen if we want to formulate user-defined predicates (e.g., `CONTAINS`) on the attribute or if we want to override DUC-consistent semantics, which is the semantics given to dimension attributes not mentioned in the flag.

A set of example queries with corresponding answers are employed to illustrate the concepts introduced above and the formulation of queries in STSQL.

Query Q1 *For each stand that is ripe in 2000, determine its harvest periods*. This query requires us to join the `stands` and the `plans` tables. We use a reducible join over the valid times to associate stands with relevant plans only. Next, we are only interested in the `stands` table as best known as of now, i.e., we restrict the transaction time to overlap *now*. This is exactly the semantics provided by DUC and we therefore do not specify any flag for `st_tt`. The location of a stand is not relevant and, thus, must be disregarded. This semantics is supported by DUC, which means that no flag for `st_area` has to be specified. Finally, we want to retrieve (and handle) the harvest periods like regular attributes. This is achieved by specifying a non-reducible flag for these dimensions.

```
> REDUCIBLE (st_vt, pl_vt) AS vt AND
  NONREDUCIBLE (harvest1, harvest2)
    SELECT st.st_ID, harvest1, harvest2
    FROM   stands st, plans pl
    WHERE  pl.st_ID = st.st_ID
    AND    pl.ripe = 2000;
```

st_ID	harvest1	harvest2	vt
st_100	1998-2000	1999-2004	1996-now
st_560	1999-2001	2001-2003	1996-now

Query Q2 *Determine pine stands and corresponding estate(s).* This query requires us to point-wise join (a) the locations of `stands` and `estates` and (b) the valid times of both tables. Because we are only interested in information as best known, we restrict the transaction times to overlap *now*. This is exactly the semantics provided by DUC, and we therefore do not specify any flag for the transaction times.

```
> REDUCIBLE (st_area, es_area) AS area AND
  REDUCIBLE (st_vt, es_vt) AS vt
    SELECT st_ID, es_ID
    FROM   stands, estates
    WHERE  specie = 'pine';
```

st_ID	es_ID	area	vt
st_100	es_80	reg_{st_100A}	1996-now
st_100	es_34	reg_{st_100C}	1995-now

Query Q3 *For all stands, determine when the two harvest periods are scheduled contemporarily.* Searching for contemporary occurrences (i.e., instants within both harvest periods) hints at reducible semantics. In this case, we have to self-join the `stands` table, relating `harvest1` and `harvest2` in a point-wise fashion. Note that a nonreducible semantics has to be specified for those harvest periods we are not interested in, i.e., `harvest2` for `pl1` and `harvest1` for `pl2`, respectively. We have to do so to prevent the default, DUC-consistent evaluation. Such an evaluation would restrict the times of the respective harvest periods to the current time, which is clearly not what we want.

```
> REDUCIBLE (pl1.harvest1, pl2.harvest2)
            AS agreed_harvest AND
  NONREDUCIBLE (pl1.harvest2, pl2.harvest1)
    SELECT pl1.st_ID
    FROM   plans pl1, plans pl2
    WHERE  pl1.st_ID = pl2.st_ID;
```

pl1.st_ID	agreed_harvest
st_100	1999-2000
st_560	2001
st_245	2010-2012

5 Conclusion and Future Research

This paper has formulated central requirements to a new dimensional DBMS aiming at addressing legacy-related concerns. The objectives are to make it possible for legacy database applications using a conventional SQL–92-based DBMS to be migrated to a dimensional DBMS without changing the application code; to make it possible to add new spatio-temporal applications without affecting the legacy applications; and to make it possible to reuse programmer expertise in SQL-92 when developing spatio-temporal applications. A spatio-temporal extension to SQL–92, termed STSQL, that provides built-in data management support for spatio-temporal data has been designed to meet the above requirements. The core of the languages and fundamental issues and concepts in its design have been explored.

No other spatio-temporal language satisfies all the requirements. Within temporal databases, two bitemporal query languages satisfy the requirements; unlike these two languages, STSQL supports an arbitrary number of temporal and spatial attributes with built-in support in the query language.

Several directions for further explorations may be identified. First, we have only described the initial design of the core of STSQL, and further formalizations of the language, beyond the informal semantics give here, are warranted. Perhaps most prominently, the semantics of spatio-temporal modifications have yet to be determined in full, and then specified. Next, we have chosen one possible and reasonable semantics for DUC statements. It appears that other semantics are possible in a multi-dimensional framework; further studies are needed to identify these and to then explore their utility. Finally, it is desirable to implement a core subset of STSQL on top of an existing DBMS, e.g., Oracle using its Spatial Data Option. This layered approach allows for relatively quick construction of a prototype that may then be used as a vehicle for experimentation with the query language design. Previously a prototype implementation of STSQL's temporal cousin, ATSQL [3], has been implemented.

ACKNOWLEDGEMENTS

The authors would like to thank Gunnar Misund and Sverre Stikbakke who introduced us to the Ecoplan forest management application. The responsibility for any simplifications or distortions of that application is ours entirely.

This research was supported in part by the Danish Technical Research Council through grant 9700780, the Norwegian Research Council through grant MOI.31297, and by the CHOROCHRONOS project, funded by the European Commission DG XII Science, Research and Development, as a Networks Activity of the Training and Mobility of Researchers Programme, contract no. FMRX-CT96-0056.

REFERENCES

[1] Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843.

[2] Bair, J., Böhlen, M. H., Jensen, C. S., and Snodgrass, R. T. 1997. Notions of Upward Compatibility of Temporal Query Languages. *Wirtschaftsinformatik*, 39(1):25–34.

[3] Böhlen, M. H., and Jensen, C. S. 1996. Seamless Integration of Time into SQL. Technical Report R-96-49, Department of Computer Science, Aalborg University.

[4] Böhlen, M. H., Jensen, C. S., and Snodgrass, R. T. 1995. Evaluating the Completeness of TSQL2. In Clifford, J. and Tuzhilin, A. (eds.), *Proceedings of the International Workshop on Temporal Databases*, pp. 153–172, Zurich.

[5] Cheng, T. S., Gadia, S. K., and Nair, S. S. 1992. Relational and Object-Oriented Parametric Databases. Technical Report TR-92-42, Computer Science Department, Iowa State University.

[6] Couclelis, H. 1992. People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. In *Lecture Notes in Computer Science*, Volume 639, pp. 65–77, Springer-Verlag.

[7] Egenhofer, M. J. 1994. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95.

[8] Egenhofer, M. J., and Franzosa, R. D. 1991. Point-set Topological Spatial Relations. *International Journal on Geographical Information systems*, 5(2):161–174.

[9] Gadia, S. K. 1988. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448.

[10] Güting, R. H., and Schneider, M. 1995. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):243–286.

[11] Ingram, K., and Phillips, W. 1987. Georgraphic Information Processing using a SQL-based Query Language. In Chrisman, N. R. (ed.), *Proceedings of the International Symposium on Computer-Assisted Cartography*, pp. 326–335, Baltimore, MD.

[12] Jensen, C. S., and Snodgrass, R. T. 1994. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974.

[13] Katz, R. H. 1990. Towards a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 20(4):375–409.

[14] Lorentzos, N. A. 1993. The Interval-extended Relational Model and Its Application to Valid-time Databases. In *Temporal Databases: Theory, Design, and Implementation*, Chapter 3. Benjamin/Cummings Publishing Company.

[15] Melton, J., and Simon A. 1993. *Understanding the New SQL: A Complete Guide*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

[16] Misund, G., Johansen, B., Hasle, G., and Haukland, J. 1995. Integration of Geographical Information Technology and Constraint Reasoning—a Promising Approach to Forest Management. Technical Report STF33A 95009, SINTEF Applied Mathematics, Oslo, Norway.

[17] Ooi, B. C., Sacks-Davis, R., and McDonell, K. J. 1989. Extending a DBMS for Geographic Applications. In *Proceedings of the Fifth IEEE International Conference on Data Engineering*, p. 590, Los Angeles, CA.

[18] Oosterom, P. v., and Vijlbrief, T. 1991. Building a GIS on Top of the Open DBMS Postgres. In *Proceedings of EGIS'91*, pp. 775–787, Brussels, Belgium.

[19] Roussopoulos, N., Faloutsos, C., and Sellis, T. 1988. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650.

[20] Snodgrass, R. T. 1990. Temporal Databases: Status and Research Directions. *ACM SIGMOD Record*, 19(4):83–89.

[21] Snodgrass, R. T. 1987. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298.

[22] Snodgrass, R. T., Böhlen, M. H., Jensen, C. S. , and Steiner, A. 1996. Adding Valid Time to SQL/Temporal. ANSI Expert's Contribution, ANSI X3H2–96–501r1, ISO/IEC JTC1/SC21/ WG3 DBL MAD–146r2, *International Organization for Standardization*.

[23] Snodgrass, R. T., Böhlen, M. H., Jensen, C. S. , and Steiner, A. 1996. Adding Transaction Time to SQL/-Temporal. ANSI Expert's Contribution, ANSI X3H2–96–502r2, ISO/IEC JTC1/SC21/WG3 DBL MCI–147r2, *International Organization for Standardization*.

[24] Snodgrass, R. T., (ed.) 1995. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers.

[25] Worboys, M. F. 1995. *GIS: A Computing Perspective*. Taylor & Francis Ltd, London.