Rapporto di ricerca
Research report

RR 83/2011

June 2011

ST4SQL: a spatio-temporal query language dealing with granularities

Gabriele Pozzani

gabriele.pozzani@univr.it

Carlo Combi

carlo.combi@univr.it

Abstract

In many different application fields the amount and importance of spatio-temporal data (i.e., temporally and/or spatially qualified data) is increasing in last years and users need new solutions for their management. In [3] a framework for modeling spatio-temporal data in relational databases has been proposed. The model is based on the notions of temporal, spatial, and spatio-temporal granularities. Once data have been modeled, they must be queried through query languages able to exploit also their spatio-temporal components. In this paper we propose a spatio-temporal query language, called ST4SQL. The proposed language extends the well-known SQL syntax and the T4SQL temporal query language [7]. The proposed query language deals with different temporal, spatial, and spatio-temporal semantics with respect to temporal and spatial dimensions. These semantics allow one to specify how the system must manage spatio-temporal dimensions for evaluating the queries. Moreover, the query language introduces new constructs for grouping data with respect to temporal and spatial dimensions. Both semantics and grouping constructs take into account and exploit data qualified with spatio-temporal granularities.

1 Introduction

Spatio-temporal databases are those databases providing support for representing and managing temporal, spatial and spatio-temporal data. They implement capabilities of both temporal and spatial databases and, further, they provide new constructs for representing spatio-temporal data, i.e., data with both a temporal and a spatial qualification.

Research on spatio-temporal databases began in the late 1990s with two different projects [19, 15]. Thenceforth, several papers [3, 4, 5, 14, 22] proposed to manage spatio-temporal data in databases by qualifying them with spatio-temporal granularities. In general, granularities represent partitions of a temporal and/or spatial domains in disjoint intervals or regions (called granules), e.g., municipalities, pollution areas. Spatio-temporal data can be temporally and/or spatially qualified associating them to the granule representing their temporal or spatial location. In [3] authors proposed the definitions of spatial and spatio-temporal granularities and, based on them and on the notion of temporal granularity [4], they proposed a relational database for granularities. The database can be used to store information about temporal, spatial and spatio-temporal granularities. Authors proposed to use the database for granularities to extend existing spatio-temporal databases in order to qualify spatio-temporal data with granularities.

Once spatio-temporal data have been modeled in relational databases, they should be queried. For this purpose we need a query language able to manage spatio-temporal data. SQL [11] includes the definition of temporal datatypes (e.g., date, timestamp, interval) and functions. However, it may be not enough. SQL considers temporal data as other classical data and users have to manage them "manually", forcing their semantics. The same approach has been applied to spatial datatypes. The ISO/IEC 13249-3 SQL/MM Part 3 [12] standard includes the definition of spatial datatypes for representing 2- and 3-dimensional geometries (e.g., points and regions) and functions for their management. However, again, spatial data are considered as other classical data. This approach, used for both temporal and spatial data, may lead to not exploit full potential of temporal and spatial dimensions. A different approach has been discussed in several proposals about temporal query languages [7, 20]. In these proposals, temporal dimensions are in some sense considered "meta-data" enriching classical information. In this way, particular meaning may be associated to temporal dimensions and this meaning can be used to better exploit temporal qualification of data. In particular, T4SQL [7] includes also the definition of four temporal semantics (i.e., atemporal, sequenced, current, and next) that can be specified by the user for the automatic management of temporal dimensions during query evaluation.

Based on temporal and spatial capabilities, some spatio-temporal query languages have been proposed [8, 9, 19, 21]. In general, these query languages define spatio-temporal datatypes and a set of functions on them. Thus, they lack of ad-hoc semantics for spatio-temporal dimensions that may help users to easily manage spatio-temporal data.

In this paper, we propose an extension of T4SQL, called ST4SQL, for supporting also spatial and spatio-temporal dimensions. ST4SQL introduces new constructs for querying data exploiting their spatio-temporal components. ST4SQL adds to T4SQL support to spatio-temporal granularities [3] and data qualified with granularities and it introduces support for spatial and spatio-temporal dimensions. In particular, we extend the four temporal semantics proposed in T4SQL also to consider data qualified with temporal granularities. Moreover, we define spatial and spatio-temporal semantics for dealing with data qualified with spatial and spatio-temporal granularities. We propose also constructs for grouping data and applying aggregate functions with respect to their temporal, spatial, and spatio-temporal qualification based on granularities. We introduce the syntax of ST4SQL and we explain the usage of new constructs exemplifying them with queries on a real clinical database.

The paper is organized as follow. In the next section we discuss main related work about spatio-temporal query languages. In Section 3 we briefly introduce granularities and the database model on which ST4SQL is based. Moreover, we introduce the clinical spatio-temporal database we then use as an example. In Section 5 we describe the ST4SQL query language, exemplifying it with several queries. In Section 6 we conclude with final remarks and future work.

2 Related work

Several temporal query languages have been proposed [7, 17, 20] to overcome the limitations of SQL with respect to temporal databases.

In 1995, Snodgrass et al. proposed TSQL2 [20], a temporal extension to the SQL-92 standard query language. Despite many researches proved the usefulness of TSQL2, the project for incorporating some TSQL2 capabilities into the ISO SQL standard has been canceled in 2001. However, some similar temporal features have been implemented in Oracle Database [16] and IBM DB2 [10].

The draft proposed for adding temporal support in SQL standard, called SQL/Temporal, includes the support for two temporal dimensions and two semantics for temporal queries. Supported temporal dimensions are valid time, i.e., the time instants or intervals when an information is true in the modeled reality, and transaction time, i.e., the time interval during which data are current and can be retrieved in the database [13]. Temporal semantics allow a user to specify how the DBMS has to (automatically) manage temporal information during the evaluation of a query. According to the sequenced semantics the query engine evaluates the query for each time instant in the time domain selecting only those tuples whose value for the considered temporal dimension includes the considered instant. Conversely, in the non-sequenced semantics the query is evaluated considering all tuples in the relations without regard to their value for the given temporal dimension.

In [7] Combi et al. proposed the T4SQL temporal query language. It extends SQL/Temporal adding support also for availability time (i.e., the time when the database system or user become aware of a fact), and event time (i.e., the time when a decision has been taken or an event happened determining the considered fact) [13]. Moreover, besides the mentioned sequenced and non-sequenced (called atemporal) semantics, T4SQL introduces other two semantics: (1) current (the DBMS evaluates the query only on those tuples where d is equal or contains the current date) and (2) next (the DBMS considers only pairs of tuples related to the same entity and that are consecutive with respect to the temporal ordering).

Both previous languages include support for temporal join and temporal grouping. The first one is an extension of natural join in which join conditions include one or more temporal dimensions. The latter allows to group tuples according to their value on a temporal dimension.

Considering spatial databases, currently, many commercial DBMSs that implement spatial functionalities, offer spatial datatypes, relationships, and operations following the approach standardized in ISO/IEC 13249-3 SQL/MM Part 3 [12]. SQL/MM is a standard extending SQL with multimedia and application-specific features. In particular, its third part defines how to store, retrieve, and process spatial data in SQL. Spatial data types and functions for converting, comparing, and analyzing spatial data are defined.

SQL/MM includes functions for testing the validity of relationships between spatial data, and computing operations over spatial objects. Spatial functions include, for example, `ST_Area` that computes the area of the geometry on which it is applied and `ST_Overlaps()` that tests whether the geometry on which it is applied overlaps the geometry provided as parameter. By using these functions spatially-enabled DBMSs allow one also to perform spatial selection (i.e., a selection operation based on a spatial predicate) and spatial join (i.e., a join operation which join condition includes a spatial predicate).

After TSQL2 and SQL/MM Part 3, despite several spatio-temporal query languages have been proposed [6, 8, 9, 19, 21], there have not been efforts for standardizing a spatio-temporal model and query language.

Erwig and Schneider [8] extend spatial functions adding a temporal components. In this way, spatial functions may be applied to moving points and regions. For example, the `trajectory()` function returns a polyline representing the trajectory of a moving point over time while `traversed` computes the region traversed in any time by a moving region.

A slightly different approach has been proposed by Sistla et al. [19]. Authors introduce a query language based on the FTL (Future Temporal Logic) language [18]. FTL queries are specified in the `Retrieve-Where` form, where the `Retrieve` clause specifies which data have to be returned and the `Where` clause specifies the FTL formula representing the condition that the retrieved data must comply with. The `NextTime` and `Until` modal temporal operators may be used in the `Where` clause. Temporal conditions can use spatial relations in order to perform spatio-temporal selection.

3 A data model supporting granularities

In [3] a framework for modeling and qualifying spatio-temporal data in relational databases using spatio-temporal granularities has been proposed. Informally, a temporal granularity represents a partition of a time domain. Each element of a granularity is called *granule* and represents a set of time instants perceived and used as an indivisible entity. Granules must be disjoint and their order must coincide with time point order. We can use these granules to provide data with a temporal qualification at the suitable granularity. In other words, a temporal granularity represents a temporal unit of measure. For example, we can associate a patient's hospitalization with the granule representing September 30, 2010, thus proving it with a temporal qualification.

Similarly, a spatial granularity represents a partition of a space domain. Granules must have disjoint interior and may have holes and may be composed by several disconnected regions. In [3] spatial granularities are defined by using a two-level model. The lower level represents the spatial domain, on which we can recognize geometrical information and in which vector data representing granules are defined. The higher level is a multidigraph structure used to access and manage granules. A multidigraph is a labeled directed graph with multiple labeled edges. In the multidigraph each node represents a spatial granule. On the other hand, edges represent relationships between granules (e.g., direction- and distance-based relations). Each edge is labeled with the name of the relationship it represents.

Based on these two frameworks, the notion of spatio-temporal granularity has been proposed in [3]. Spatio-temporal granularities represent the evolution over time of a spatial granularity. A spatio-temporal granularity has two components. The former is a temporal granularity that aggregates time points, while the latter is a mapping (called *spatial evolution*) that associates to each time point the spatial granularity valid on it.

In [3] a database for granularities has been designed. The database allows one to store all information describing temporal, spatial, and spatio-temporal granularities. The following is a part of the relational schema of the database for granularities.

```
GRANULE(id,index,start,end,granularity)
T-GRANULARITY(id,name,domain,extentS,
              extentE,anchor)
ST-GRANULARITY(id,name,tgranularity,evolution)
EVOLUTION(id,name)
VALIDTIMEHISTORY(evolution,sgranularity,since,to)
S-GRANULARITY(id,name,domain,extent,domain_rel)
NODE(id,sgranularity,geom,label)
NODE-LABEL(id,label)
```

where T-GRANULARITY and GRANULE represent temporal granularities and their granules, respectively. S-GRANULARITY contains information about spatial granularities, while NODE and NODE-LABEL represent spatial granules (the nodes of the multidigraph) and their labels, respectively. Finally, ST-GRANULARITY, EVOLUTION, and S-GRANULARITY represent spatio-temporal granularities and their spatial evolutions.

4 A motivating clinical database

The designed database for granularities may be used to qualify and aggregate data, also during the querying phase, in an existing relational database. In [3] authors propose two ways to qualify and aggregate information by using granularities. The first way provides to add, in each table containing data we want to qualify, a reference (i.e., a foreign key) to the granule representing the temporal or spatial location. The second way allows one to qualify data with granularities directly during the querying phase. In this case, tuples we are interested in are associated to a temporal and/or a spatial location (e.g., a timestamp or a coordinate on the Earth surface). Later (e.g., in a query) it is possible to "dynamically" aggregate information with respect to the temporal or spatial locations we added by using suitable granularities.

Let us consider for example the following database schema:

```
PATIENT(id,tax_code,surname,name,birth_date)
CONTACT(id,patient,contact_date, gaf
        contact_location, duration)
PATIENT_DIAGNOSIS(id,patient,diagnosis,
                  start_date,end_date)
DIAGNOSIS(id,category,icd10_code,
          description)
PATIENT_PROFESSION(id,patient,profession,
                   start_date,end_date)
PROFESSION(id,description)
PATIENT_EMPLOYMENT(id,patient,employment,
                   start_date,end_date)
EMPLOYMENT(id,description)
PATIENT_RESIDENCE(id,patient,address,
                  subzone,start_date,end_date)
PATIENT_DOMICILE(id,patient,address,subzone,
                 start_date,end_date,location)
CONTACT_LOCATION(id,description,location)
SUBZONE(id,description,quarter,municipality)
MUNICIPALITY(istat_code,name,inhabitants_2001,
             node)
```

It is a part of the schema of the Verona Psychiatric Case Register (PCR). Verona is a city in North-East of Italy. Its Health District includes a Community-based Psychiatric Service (CPS) for providing support to psychiatric patients. PCR is the information system collecting information about patients' accesses to CPS since 1979 [1]. At the first contact with the psychiatric service (e.g., a visit), socio-demographic information, past psychiatric and medical history, and clinical data are routinely collected for patients aged 14 years and over. These data may be updated at successive contacts when required. Each patients' contact with CPS structures is recorded in PCR. Recorded contacts include: attendances at the out-patient clinic, domiciliary visits, telephone calls, day cares provided at the day hospital units, all admissions to the acute psychiatric ward, and private clinics. Data on about 28,700 patients and more than 1,500,000 psychiatric contacts have been recorded in PCR up to now.

To each patient a diagnosis is assigned according to ICD-10 [23] categories. Moreover, PCR stores also patients' employment and profession. All these data may be updated when required, and thus PCR records also their valid time.

PCR is currently used for administrative, clinical, and research purposes.

The relations of PCR we reported above exemplify the two ways for using granularities for aggregating and querying spatio-temporal data. The relation `Municipality` contains a reference (i.e., the `node` attribute) to the corresponding granule in the `Municipalities` spatial granularity. On the other hand, the `Patient_domicile` relation has a `location` attribute representing the coordinates on the Earth surface where the domicile is located. Moreover, all temporal relations contain the `start_date` and `end_date` attributes representing the valid time of the tuples in the relations. `location` and valid time attributes may be used for spatially and/or temporally aggregating data at the querying phase.

In the following we will exemplify the capabilities of our ST4SQL query language also presenting some queries on PCR. These queries refer to patients' personal information and contacts with CPS and they may be useful to get reports for administrative purposes.

5 The ST4SQL query language

In this section, we present our query language for dealing with spatio-temporal data and granularities. It is a spatio-temporal extension of T4SQL, the temporal query language proposed by Combi et al. [7] and discussed in Section 2. Thus, ST4SQL is an SQL-based query language. It extends the temporal constructs of T4SQL to deal with temporal granularities and it adds new spatial and spatio-temporal constructs, similar to the temporal ones, for dealing also with spatial and spatio-temporal granularities.

ST4SQL queries are based on the usual SELECT-FROM-WHERE structure, but ST4SQL adds new constructs for the "automatic" management of temporal, spatial, and spatio-temporal dimensions. Supported temporal dimensions include valid time, transaction time, availability time, and event time [13]. On the other hand, in the spatial case we consider only valid space, that is the spatial location where an object instance is "true". Values for valid space can be modeled, for example, as points, lines, or regions.

As we already explained, one of the main contributions of T4SQL has been the introduction of four semantics for temporal queries. These semantics allows one to specify how the system has to manage temporal dimensions in order to evaluate queries. The proposed semantics for temporal querying are defined on time points. Thus, for example, the SEQUENCED semantics applied to valid time considers instant by instant the temporal domain and evaluates the query taking into account only those tuples whose valid time contains the considered time point.

However, we can consider semantics also on temporal granularities. In this way, the system evaluates queries considering granules instead of time points and, thus, constraining tuples with respect to their value on temporal dimensions according to temporal granules. To do that, we extend previous temporal semantics. Thus, each semantics requires the user to specify a time dimension (e.g., valid time) and a temporal granularity (e.g., months). In other words, semantics based on granularities allow one to "relax" T4SQL queries by grouping tuples with respect to the granules of a temporal granularity instead of time points.

Similar definitions can be given also on spatial dimensions. A spatial semantics allows one to specify how the query engine must manage spatial dimensions associated to tuples. Similarly to the temporal case, we define four spatial semantics: SPACELESS (corresponding to the ATEMPORAL one), SEQUENCED, CURRENT, and NEXT.

Note that we define only spatial semantics based on granularities, not on space points. This limitation is based on the consideration that joining only tuples spatially qualified exactly with the same point location is not meaningful, especially considering that many errors and uncertainties can affect spatial surveys.

Considering spatio-temporal granularities, we note that they temporally qualify spatial granularities. Thus, they allow one to spatially aggregate data with respect to a spatial granularity whose definition changes over time. The value of tuples over a given temporal dimension is used to select the spatial granularity valid at the same time of considered tuples. Thus, a spatio-temporal semantics requires that both a temporal dimension and a spatial dimension are specified, we call this pair a spatio-temporal dimension. On this pair, the user specifies a semantics on both the temporal and the spatial dimension. Then, the temporal and spatial semantics are applied as we have described before in this section, but, moreover, the temporal dimension is used also for selecting the spatial granularity valid at the same time of considered tuples.

The syntax of ST4SQL extends that of T4SQL (and that of SQL). Its (incomplete) BNF is as follows:

```
[SEMANTICS [<tsem> ON <tdim>
    [WITH TGRANULARITY <tG>] [THROUGH <attr>]
    [TIMESLICE <ts_exp>] [, ...],]
    [<ssem> ON <sdim> WITH SGRANULARITY <sG>
    [ORDERED BY <sr>] [THROUGH <attr>]
    [SPACESLICE <ss_exp>] [, ...],]
    [<tsem> ON <tdim> [TIMESLICE <ts_exp>] AND
    <ssem> ON <sdim> [SPACESLICE <ss_exp>]
    WITH STGRANULARITY <stG>
    [SIMPLIFY BY <s_aggr_funct>]]]
SELECT <sel_element_list>
    [WITH <exp> [AS] <dim> [, ...]]
    [, TGROUP(<t_attr>) [AS] <new_name> [,...]]
    [, SGROUP(<s_attr>) [AS] <new_name> [,...]]
[FROM <tables>]
[WHERE <conditions>]
[WHEN <t_conditions>]
[WHEREABOUTS <s_conditions>]
[GROUP BY <group_element_list>]
[HAVING <g_cond>]

<tsem>::= ATEMPORAL | CURRENT | SEQUENCED | NEXT
<ssem>::= SPACELESS | CURRENT | SEQUENCED | NEXT
<dim> ::= <tdim> | <sdim>
<tdim>::= AVAILABILITY | TRANSACTION |
    VALID TIME | INITIATING_ET | TERMINATING_ET
<sdim>::= VALID SPACE
<group_element_list>::= <group_element> [, ...]
<group_element>::= <attribute> |
    <temp_attr> ON <tG> | <space_attr> ON <sG>
```

where:

- tG and sG represent a temporal and a spatial granularity, respectively;
- ts_exp and ss_exp represent expressions corresponding to a time interval and a spatial region, respectively;
- <sr> is the name of a spatial relationship between granules whose definition must be present in the SR relation of the proposed database for granularities.

We introduce also some spatio-temporal functions for accessing the value of tuples on temporal and spatial dimensions. Given a tuple of a relation R, the value it takes over the temporal and the spatial dimensions can be recovered by using the following accessors:

VALIDT(R) returns the valid time of the R tuple;
TRANSACTIONT(R) returns the transaction time of the R tuple;
AVAILABLET(R) returns the availability time of the R tuple;
INITIATING_ET(R) returns the initiating event time of the R tuple;
TERMINATING_ET(R) returns the terminating event time of the R tuple;
VALIDS(R) returns the valid space of the R tuple;

As we already mentioned in previous section, Belussi et at. proposed two ways to qualify data with spatio-temporal granularities in two alternative ways: (1) indirect (tuples are associated to a temporal and/or spatial location that is qualified with granularities at query time) and (2) direct (tuples are associated directly to a granule in a temporal, spatial, or spatio-temporal granularity). In the first case accessors return the location associated to the tuples, while in the second case accessors return the granule associated to the tuples.

Finally, considering any expression <exp> representing an interval (e.g., VALIDT(R), ts_exp) we access its starting and ending element with <exp>$start and <exp>$end, respectively.

Since ST4SQL includes SQL and T4SQL, we now describe only new constructs defined in ST4SQL. For this reason, considering temporal semantics, we describe only their new versions based on temporal granularity.

5.1 The clause SEMANTICS

The SEMANTICS clause allows one to specify the semantics to be applied for temporal, spatial, or spatio-temporal dimension. More semantics may be specified on different dimensions, while at most one semantics can be applied to a dimension, otherwise the query is considered to be not well-formed.

In particular, a semantics for a temporal dimension may be specified with the following syntax:

```
SEMANTICS <tsem> ON <tdim>
   [WITH TGRANULARITY <tG>] [TIMESLICE <ts_exp>]
```

while, with a similar syntax, the user may specify a spatial semantics:

```
SEMANTICS <ssem> ON <sdim>
   WITH SGRANULARITY <sG> [ORDERED BY <sr>]
   [THROUGH <attr>] [SPACESLICE <ss_exp>]
```

and a spatio-temporal semantics:

```
SEMANTICS <tsem> ON <tdim> [TIMESLICE <ts_exp>]
   AND <ssem> ON <sdim> [SPACESLICE <ss_exp>]
   WITH STGRANULARITY <stG>
   [SIMPLIFY BY <s_aggr_funct>]
```

As we have already explained above, each semantics requires a dimension (<tdim> or <sdim>) on which it has to be applied. Moreover, optionally, a temporal semantics can be applied in relation to a temporal granularity <tG> that will be used to group and select the tuples on which the query has to be evaluated. Since spatial semantics are defined only on granularities, the WITH SGRANULARITY <sG> token is mandatory and <sG> is the name of the spatial granularity to be used.

The TIMESLICE and SPACESLICE options are optional and allow one to restrict the query evaluation only on those tuples whose value for the given temporal or spatial dimension intersects the value of <ts_exp> and <ss_exp> expressions, respectively. <ts_exp> must represent a time interval, when the WITH TGRANULARITY <tG> option is not specified, or to a pair (gstart,gend) representing a granule interval where gstart and gend are the indexes of the first and the last granule in <tG> in the interval, when the WITH TGRANULARITY <tG> token is present. Similarly, <ss_exp> is a list of labels of granules in the <sG> spatial granularity. The TIMESLICE and SPACESLICE options can be specified together with any semantics but the CURRENT one.

Note that, when no semantics is specified by the user on a temporal or spatial dimension, the ATEMPORAL or SPACELESS semantics (depending on the dimension) is applied as default by the system on that dimension.

5.1.1 ATEMPORAL temporal semantics

The ATEMPORAL semantics disables any system support on the specified temporal dimension. Thus, the given dimension is considered as a usual atemporal attribute without any specific meaning. In this case, unless other spatio-temporal constructs are specified, queries are evaluated as usual SQL statements and the user is free to manage the temporal dimension whenever she needs.

With the ATEMPORAL semantics, the TIMESLICE <ts_exp> and WITH TGRANULARITY <tG> options can be specified together in order to limit the query evaluation only to some granules of <tG>. <ts_exp> is a pair (gstart,gend) representing a granule interval where gstart and gend are the indexes of the first and the last granule in <tG> in the interval, respectively.

Thus, for example, the following query retrieves the patients who in 2009 had at least a contact lasted more than one hour. In this case, the valid time is represented by the date when contacts occurred.

```
SEMANTICS ATEMPORAL ON VALID TIME
   WITH TGRANULARITY Years TIMESLICE (2009,2010)
SELECT DISTINCT p.name, p.surname
FROM patient AS p JOIN contact AS c
     ON c.patient=p.id
WHERE c.duration > '01:00:00'
```

The output relation does not include the dimension on which the ATEMPORAL semantics has been applied. The user can include the temporal dimension in the output relation by specifying the WITH option in the SELECT clause (discussed below).

Note that, since the ATEMPORAL semantics disables any automatic support leaving the user free to manage also the temporal dimensions, one can execute an ATEMPORAL query to obtain the same result that could be obtained by applying any other semantics.

5.1.2 SEQUENCED temporal semantics

Using the SEQUENCED semantics the DBMS evaluates the query only on those tuples of the relations referred in the FROM clause whose value for the given temporal dimension intersects the same temporal granule of the temporal granularity

`<tG>`. In other words, the query is evaluated granule by granule considering in turn only the tuples whose value on the provided temporal dimension intersects the considered granule. Thus, the output relation is a temporal relation where the query result is associated to the granule on which it has been evaluated.

For example, the following query allows us to retrieve monthly the average number of contacts per patient for unemployed patients.

```
SEMANTICS SEQUENCED ON VALID TIME
          WITH TGRANULARITY Months
SELECT COUNT(c.id)/COUNT(DISTINCT c.patient)
FROM contact AS c, patient_employment AS pe,
     employment AS e
WHERE c.patient = pe.patient AND
      pe.employment = e.id AND
      e.description = 'unemployed'
```

Since both `contact` and `patient_employment` relations support valid time, the `SEQUENCED` semantics automatically restricts the query evaluation only on those contacts occurred in the same month in which the corresponding patient declared, for at least a day, to be unemployed. Moreover, since the `SEQUENCED` semantics provides that the considered granule is included in the output relation, the query returns a relation that includes also the month in which that data are valid.

5.1.3 `CURRENT` temporal semantics

The `CURRENT` semantics forces the DBMS to evaluate the query only on those tuples whose value over the given temporal dimension intersects the granule of `<tG>` containing the current time. In other words, the `CURRENT` semantics is similar to the `SEQUENCED` one but it does not evaluate the query for each granule in the specified granularity but only in the "current" granule.

For example, by using the `CURRENT` semantics we can retrieve the patients' professions in the current month. Since the query is evaluated on those tuples whose valid time intersects the current month, if a patient changes her profession during the current month all her professions are returned.

```
SEMANTICS CURRENT ON VALID TIME
          WITH TGRANULARITY Months
SELECT p.name, p.surname, pr.description
FROM patient AS p, patient_profession AS pp,
     profession AS pr
WHERE pp.patient=p.id AND pp.profession=pr.id
```

Conversely to the `SEQUENCED` semantics, the `CURRENT` semantics does not include in the output relation the granule on which the query is evaluated. Thus, unless other spatio-temporal constructs are specified by the user, the output relation is atemporal.

5.1.4 `NEXT` temporal semantics

The `NEXT` semantics allows one to retrieve information about the state of the same object in two consecutive granules. In other words, the `NEXT` semantics considers only pairs of tuples (each one belonging to the join of the tables specified in the FROM clause) that are related to the same object and that are in two consecutive granules of <tG> according to the temporal ordering.

When the FROM clause contains more than one relation, they are joined using join conditions specified by the user in the query or Cartesian product when join conditions are not specified. Tuples in the relation resulting from this join are used for defining the pairs on which the query has to be evaluated.

Two tuples are related to the same object if they have the same atemporal key. If atemporal keys are not defined or more keys are present in relations referred in the FROM clause, the user must specify the attribute to use for identifying tuples referring to the same object. To do that, the THROUGH <attr> option may be specified, and the <attr> attribute is used instead of snapshot keys for joining pairs of tuples. With the NEXT semantics, the user can use the NEXT(<attr>) function to refer, in the select and where clauses, to the value of a <attr> attribute in the tuple associated to the successor granule.

For example, the following query retrieves the patients who found a new job after a period of unemployment and who, in that month, had a worse Global Assessment of Functioning (GAF) value [2], i.e., patients whose mental health seems to get worse with the new job.

```
SEMANTICS NEXT ON VALID TIME
   WITH TGRANULARITY Months THROUGH patient.id
SELECT DISTINCT p.name, p.surname
FROM patient AS p, patient_employment AS pe,
     employment AS e, contact AS c
WHERE pe.patient=p.id AND pe.employment=e.id AND
      c.patient=p.id AND NEXT(c.gaf)<c.gaf AND
      e.description = 'unemployed' AND
      NEXT(e.description) <> 'unemployed'
```

5.1.5 SPACELESS spatial semantics

The SPACELESS spatial semantics corresponds to the ATEMPORAL one we described before. The SPACELESS semantics disables any support from the system and the user has, eventually, to manage explicitly the spatial dimension on which it is applied. No spatial granularity has to be specified together with the SPACELESS semantics unless the SPACESLICE option is specified: in this case the granularity is mandatory.

For example, the following query retrieves the patients living in a highly polluted area and contacting CPS structures located in little polluted areas.

```
SEMANTICS SEQUENCED ON VALID TIME
   WITH TGRANULARITY Weeks,
   SPACELESS ON VALID SPACE
SELECT DISTINCT p.tax_code, p.name, p.surname
FROM patient AS p JOIN contact AS c
          ON p.id=c.patient
     JOIN patient_domicile AS pd
          ON pd.patient = p.id
     JOIN contact_location AS cl
          ON c.contact_location = cl.id,
     s-granularity AS sg JOIN node AS n1
          ON n1.sgran = sg.id
     JOIN nodelabel AS nl1 ON n1.label = nl1.id
     JOIN node AS n2 ON n2.sgran = sg.id
     JOIN nodelabel AS nl2 ON n2.label = nl2.id
WHERE sg.name = 'PM10' AND nl1.label = 'High'
   AND nl2.label = 'Low' AND
   n1.geom.st_contains(pd.location) AND
   n2.geom.st_contains(cl.location)
```

Where `st_contains` is the SQL/MM procedure that checks whether the object on which it is called contains the geometry provided as a parameter.

Using the `SPACELESS` semantics, the user disable the system support on the spatial dimension and she manage spatial data explicitly. For this reason, spaceless queries may be longer, more complex, and less readable than queries based on other semantics.

The output relation does not include the dimension on which the `SPACELESS` semantics has been applied.

Since the `SPACELESS` semantics disables any automatic behavior, it is possible to write a spaceless query in which spatial dimensions are explicitly managed in the way that it has the same result that could be obtained by applying any other spatial semantics.

5.1.6 `SEQUENCED` spatial semantics

The `SEQUENCED` semantics evaluates the query granule by granule. For each granule, only those tuples whose value for the spatial dimension intersects the considered spatial granule are taken into account.

It allows us to retrieve, for example, the final report of contacts of patients living (when contacts occurred) in the same subzone (i.e., quarter) where contacts took place.

```
SEMANTICS SEQUENCED ON VALID TIME,
          SEQUENCED ON VALID SPACE
          WITH SGRANULARITY Subzones
SELECT c.patient, c.report
FROM patient_domicile AS pdom, contact AS c,
     contact_location AS cl
WHERE c.patient = pdom.patient AND
      c.contact_location = cl.id
```

The `SEQUENCED` temporal semantics restricts the query evaluation to the patients' domiciles valid when contacts occurred. On the other hand, the

SEQUENCED spatial semantics provides that contacts occur in the same subzone where patient's domicile is located.

Similarly to what happens in the temporal case, the SEQUENCED semantics provides that the spatial dimension on which it is applied is included in the output relation. Thus, in the previous query both the time instant and the spatial granule are returned besides other required data.

Considering another example, the following query returns the patients whose residence is currently in a municipality that intersects an area with a high level of acoustic noise.

```
SEMANTICS CURRENT ON VALID TIME,
   SEQUENCED ON VALID SPACE
   WITH SGRANULARITY DayAcousticNoise
   SPACESLICE 'High'
SELECT p.tax_code, p.name, p.surname
FROM patient AS p, patient_residence AS pr,
     subzone AS s, municipality AS m
WHERE pr.patient = p.id AND
      pr.subzone = s.id AND
      s.municipality = m.istat_code
```

The SEQUENCED spatial semantics relates the patient's residence (through the subzone and municipality relations) with spatial granules representing the day acoustic noise levels. The SPACESLICE token has been used to restrict the query evaluation only to the spatial granule labeled with High, that we suppose to represent areas with a high level of noise pollution.

5.1.7 CURRENT spatial semantics

The CURRENT spatial semantics can be used to restrict the query evaluation only on those tuples whose value over the specified spatial dimension intersects the spatial granule containing the current geographical position of the query submitter. This information can be locally saved, for example, in a system configuration file, or can be inferred from the DBMS server machine, for example analyzing the IP address from which the query has been received.

Considering the PCR database, the CURRENT semantics can be used, for example, by a physician for retrieving diagnoses only of patients currently living in the catchment area where she works.

```
SEMANTICS CURRENT ON VALID TIME,
   CURRENT ON VALID SPACE
   WITH SGRANULARITY CatchmentAreas
SELECT DISTINCT p.name, p.surname, d.icd10_code
FROM patient_domicile AS dom, patient AS p,
     patient_diagnosis AS pdia, diagnosis AS d
WHERE dom.patient = p.id AND p.id = pdia.patient
      AND pdia.diagnosis = d.id
```

Similarly to the CURRENT temporal semantics, the spatial one cannot be paired with the SPACESLICE option and the spatial dimension is not included in the output relation.

5.1.8 NEXT spatial semantics

Similarly to the temporal one, the NEXT spatial semantics allows one to relate tuples related to the same object in two consecutive granules. As in the temporal case, two tuples are considered to be related to the same object if they share the same value on a attribute provided by the user with the THROUGH token. On the other hand, conversely to the temporal case, a unique order in not defined between spatial points and granules. For this reason, the user must provide also the spatial relationship the system has to use for ordering spatial granules and thus to find consecutive granules. This can be done with the ORDERED BY <sr> token, where <sr> is the name of a spatial relationship defined in the SR table of the database for granularities.

For example, the following query relates the number of contacts for each patient in a quarter and in the quarters that are South of it.

```
SEMANTICS NEXT ON VALID SPACE
          WITH SGRANULARITY Quarters
          ORDERED BY South THROUGH c.patient
SELECT c.patient , COUNT(c.id), COUNT(NEXT(c.id))
FROM contact AS c JOIN contact_location AS cl
         ON c.contact_location = cl.id
GROUP BY c.patient
```

The behavior of the NEXT spatial semantics is similar to the temporal one. The query is evaluated on pairs of tuples belonging to the relation resulting from the join between two instances of the relations in the FROM clause. These two instances are joined on the basis of the attribute specified in the THROUGH token. Among the obtained pairs of tuples the system selects only those whose values over the given spatial dimension intersect two consecutive granules according to the order defined by the relationship specified in the ORDERED BY token. In the SELECT and WHERE clauses, the NEXT(<attr>) function can be used to refer to the value of the <attr> attribute in the tuple associated to the successor granule.

By default, the spatial dimension on which the NEXT semantics is applied is not included in the output relation.

5.1.9 Spatio-temporal semantics

In previous examples about the usage of spatial granularities, we always aggregated patients' domiciles using a unique Municipality granularity, supposing that this granularity was always valid, i.e., it never changes over time. However, patients' domiciles are spatio-temporal data, thus they may be aggregated by using the spatial granularity that actually was valid at the time when domiciles were valid. Spatio-temporal granularities can be used in queries for this purpose. As a matter of fact, spatio-temporal granularities temporally qualify spatial granularities. Thus, they allow one to spatially aggregate data with respect to a spatial granularity whose definition changes over time. The value of tuples over a given temporal dimension is used to select the spatial granularity valid at the same time of considered tuples. For this reason spatio-temporal semantics require that both a temporal and a spatial dimension are specified, and we call this pair a spatio-temporal dimension.

A spatio-temporal granularity groups spatial granularities with respect to granules of a temporal granularity. Each tuple is spatially qualified by using spatial granularities valid during the time granules intersecting the value of the tuple over the given temporal dimension. Since this value may be an interval and during this interval the spatial granularity may evolve, several spatial granularities may be used to qualify each tuple. For this reason the user may specify an aggregate function to apply to spatial granularities in order to obtain just one spatial granularity among all granularities valid during the interval. Aggregate functions include, for example, `first` (that returns the spatial granularity valid at the first instant in the interval), `last` (that returns the spatial granularity valid at the last instant in the interval), and operations for calculating union, intersection, and difference of spatial granularities [3]. If the aggregate function is not provided by the user, the system uses all spatial granularities valid in the interval, one by one, for querying the considered tuple, and all these combinations are used for obtaining the final resulting relation.

The user specifies a semantics on both the temporal and the spatial dimension making up the given spatio-temporal dimension. These temporal and spatial semantics are applied as we described above in this section, but, moreover, the temporal dimension is used also for selecting the spatial granularity valid at the same time of considered tuples.

We note that specifying the `SEQUENCED` semantics on the spatial dimension and the `CURRENT` semantics (or a timesliced `SEQUENCED` semantics) on the temporal one, a user can query data with respect to a spatially qualified timeslice. On the other hand, specifying the `SEQUENCED` semantics on the temporal dimension and the `CURRENT` semantics (or a spacesliced `SEQUENCED` semantics) on the spatial one, a user can query data with respect to the evolution over time of spatial granules. The `ATEMPORAL` and `SPACELESS` semantics cannot be used in spatio-temporal semantics.

For example, the following query retrieves information about patients' domicile valid in 2010 and located, during this interval, in an area with high or medium PM_{10} pollution level.

```
SEMANTICS SEQUENCED ON VALID TIME
    TIMESLICE (2010-01-01,2010-12-31) AND
    SEQUENCED ON VALID SPACE
              SPACESLICE 'High','Medium'
    WITH STGRANULARITY PM10_Weeks
SELECT DISTINCT p.tax_code, p.name, p.surname
FROM patient_domicile AS pd, patient AS p
WHERE pd.patient = p.id
```

`PM10_Weeks` is a spatio-temporal granularity representing daily PM_{10} surveys associated to the `Weeks` temporal granularity. Thus, the query is evaluated on those tuples whose value over the valid space intersects a PM_{10} granule valid in a week intersecting also the tuples value over the valid time. The output relation includes both the temporal and spatial granules qualifying tuples.

5.2 The clause SELECT

After the `SEMANTICS` clause, ST4SQL queries continue with the usual `SELECT` and `FROM` clauses. The `SELECT` clause specifies what attributes (or expressions) have to be returned for each tuple in the resulting relation, while the FROM clause contains the list of relations (that, eventually, may be the result of a subquery) required for processing the query.

As we explained before, the dimensions on which ATEMPORAL, SPACELESS, CURRENT, and NEXT semantics are applied are not included in the output relation, while the dimensions on which the SEQUENCED semantics is applied are included in the output relation. In order to change this behavior and to include dimensions in the output relations, the user can, besides classical attributes, use the WITH token in the SELECT clause. This option overwrites the dimension that would be included automatically by the system in the output relation.

Thus, for example, the following query retrieves the duration of contacts occurred in the current month for patients born before 1950. The query includes in the output relation the contact date.

```
SEMANTICS CURRENT ON VALID TIME
          WITH TGRANULARITY Months
SELECT c.duration WITH VALIDT(c) AS VALID TIME
FROM contact AS c JOIN patient AS p
     ON c.patient = p.id
WHERE p.birth_date < '01-01-1950'
```

5.3 The clauses WHERE, WHEN, and WHEREABOUTS

The WHERE clause contains the selection and join conditions to apply in order to select the set of tuples on which the query must be evaluated. In ST4SQL, the WHERE clause may include temporal and spatial conditions (i.e., conditions involving the value of temporal and spatial dimensions). In some cases, temporal and spatial conditions are complex and for the sake of clarity they could be kept divided from classical conditions. For this purpose, ST4SQL provides the WHEN and WHEREABOUTS clauses where temporal and spatial conditions can be specified, respectively.

Let us consider for example the following query. It retrieves the duration of the contacts occurred at most one month after that the diagnosis of the corresponding patient has been established and at most at 1 km from the patient's domicile.

```
SEMANTICS SEQUENCED ON VALID TIME,
    SPACELESS ON VALID SPACE
SELECT c.duration
FROM contact AS c JOIN patient_diagnosis AS pd
     ON c.patient = pd.patient
     JOIN contact_location AS cl
     ON c.location = cl.id
     JOIN patient_domicile AS pdom
     ON c.patient = pdom.patient
WHEN VALIDT(c)$start - VALIDT(pd)$start
                    < INTERVAL '1' MONTH
WHEREABOUTS VALIDS(cl).st_distance(VALIDS(pdom))
                    < 1 KM
```

5.4 The clause GROUP BY

The GROUP BY clause allows one to define groups of tuples on which aggregate functions may be applied. ST4SQL allows the user to group tuples with respect to the their value on a temporal or spatial dimension. To do that the user can use the spatio-temporal accessors we introduced above in the GROUP BY clause. For example, GROUP BY VALIDT(t) specifies that tuples have to be grouped with respect to the valid time of the t relation. Moreover, the accessors may be followed by the ON <G> option that specifies that the groups are not defined according to the value returned by the considered accessor but by the granules of the <G> granularity, whose type (temporal or spatial) must agree with the type of the accessor. For example, GROUP BY VALIDT(t) ON <tG> groups together tuples whose valid time intersect the same granule in <tG>.

In order to include the value of groups on a dimension in the output relation, the user can use the TGROUP and SGROUP functions in the SELECT clause. These functions require as parameter the accessor used in the GROUP BY clause and return the value of the accessor if the ON <G> option has be not specified in the GROUP BY or the granule in <G> if the ON <G> option has be specified. Thus, for example:

```
SELECT TGROUP(VALIDT(T))
FROM T
GROUP BY VALIDT(T) ON <tG>
```

returns a relation where each tuple represents a group of tuples in t. Each group contains the tuples whose value over the valid time dimension intersects the same granule of <tG>. The identifier of the granule (returned by TGROUP(VALIDT(t))) used for grouping together the tuples is included the output relation.

For example, the following query retrieves in each month the average number of contacts per patient with respect to diagnosed mental disease.

```
SELECT TGROUP(VALIDT(c)), d.category,
       COUNT(c.id)/COUNT(DISTINCT c.patient)
FROM contact AS c, patient_diagnosis AS pd,
     diagnosis AS d
WHERE c.patient = pd.patient AND
      pd.diagnosis = d.id
GROUP BY VALIDT(c) ON Months, d.category
```

Similarly, also spatial dimensions and granularities can be used also for grouping tuples. The following query retrieves the average number of contacts per patient grouped with respect to the diagnosis category and the day acoustic noise level.

```
SELECT SGROUP(VALIDS(cl)), d.category,
       COUNT(c.id)/COUNT(DISTINCT c.patient)
FROM contact AS c, patient_diagnosis AS pd,
     diagnosis AS d, contact_location AS cl
WHERE c.patient = pd.patient AND
      pd.diagnosis = d.id AND
      c.contact_location = cl.id
GROUP BY VALIDS(cl) ON DayAcousticNoise,
         d.category
```

The HAVING clause allows one to specify conditions on the groups. The HAVING clause provides conditions that have to be applied to each group of tuples, then only groups that meet such conditions are retrieved in the output relation. The HAVING clause may include aggregate functions, temporal and spatial conditions.

5.5 On the semantics of ST4SQL

All ST4SQL queries can be translated into equivalent SQL queries by using standard constructs. Hence, it is possible to specify the semantics of ST4SQL with respect to the one of SQL. This can be done by showing how ST4SQL constructs can be translated into SQL. Due to the lack of space, here we show only a fragment of the ST4SQL semantics. All other cases are treated similarly to the showed one. In particular, we introduce only the translation of the NEXT spatial semantics.

Consider the following ST4SQL query:

```
SEMANTICS NEXT ON <sdim> WITH SGRANULARITY <sG>
          ORDERED BY <sr> THROUGH <attr>
SELECT <attr_list>
FROM T1, ..., Tn
WHERE <jconds> AND <sconds>
```

where <attr_list> is a list of names of attributes in the $T_1, \ldots, T_n$ relations, while <jconds> and <sconds> are the sets of join and selection conditions. Note that, the distinction between join and selection conditions is not specified by the user, but can be easily computed by the system. Moreover, we remember that when the NEXT semantics is specified, also the ORDERED BY and THROUGH options are mandatory.

The NEXT semantics allows one to evaluate a query on the pairs of tuples representing two states of an object in two consecutive granules. To do that, the first thing the system has to do is to calculate the table containing the tuples representing the objects to pair. This table is the result of the join of all relations in the original FROM clause (i.e., T1, ..., Tn) according to the join conditions <jconds>. Thus, instance is calculated as follow:

```
(SELECT
 FROM T1, ..., Tn
 WHERE <jconds>
) AS instance
```

where, as usual, when the user does not specify the join condition between all relations, the Cartesian product is applied.

Then, in the FROM clause of the translated SQL query we have to:

- join two copies of the instance relation according to their values on the <attr> attribute, provided by the user as parameter of the THROUGH option;
- add the relations containing the information about the <sG> spatial granularity.

In the WHERE clause, we have to select only the pairs whose two constituent tuples intersect two consecutive spatial granules in <sG> with respect to the order defined by the <sr> spatial relation.

When the `NEXT` semantics is applied, the user can refer in the `SELECT` and `WHERE` clauses to the value of an attribute in the successor tuple by using the `NEXT(<attr>)` function. That is translated in the SQL query as `tnext.<attr>`, while all other attributes are referred as `t.<attr>`.

Finally, the `SELECT` clause of the resulting SQL query contains all the attributes named by the user in the original ST4SQL query, where, eventually, the `NEXT(<attr>)` function is translated as explained.

Concluding, the SQL query equivalent to the considered one is:

```
SELECT <attr_list>
FROM sgranularity AS sg JOIN node AS n
          ON (n.granularity = sg.id)
     JOIN node AS nnext
          ON (nnext.granularity = sg.id),
     instance AS t JOIN instance AS tnext
          ON t.<attr> = tnext.<attr>
WHERE <sconds> AND sg.name = <sG> AND
      n.geom.st_intersects(VALIDS(t)) AND
      nnext.geom.st_intersects(VALIDS(tnext))
      AND NOT EXISTS (SELECT node.id FROM node
        WHERE node.granularity = sg.id AND
           evaluate(<sr>,n.geom,node.geom) AND
           evaluate(<sr>,node.geom,nnext.geom))
```

where `evaluate(r,g1,g2)` is a function we defined for evaluating the spatial relationship `r` between geometries `g1` and `g2` while `st_intersects` is the SQL/MM function testing whether two geometries intersect each other.

It is clear that SQL queries equivalent to ST4SQL ones are more complex, less readable, and they contain, in some cases, subqueries. Thus, computational complexity and performance issues arise. In order to keep query evaluation time reasonable, we need to take into account these problems and apply query optimization techniques and indexing structures when we will completely implement our language.

6 Conclusions

In this paper, we proposed a new query language, called ST4SQL, for dealing with temporal, spatial, and spatio-temporal dimensions qualified with granularities. ST4SQL provides support for valid time, transaction time, availability time, event time, and valid space. Moreover, it defines spatio-temporal dimensions as a combination of temporal and spatial dimensions. ST4SQL introduces four temporal and spatial semantics that, combined, define spatio-temporal semantics. Semantics add a specific meaning to queries and allow the user to specify how the system has to (automatically) manage dimensions for the query evaluation. Moreover, ST4SQL allows the user to group data with respect to their spatio-temporal components. In ST4SQL all proposed constructs may use granularities for qualifying and querying spatio-temporal data. We introduced the syntax of ST4SQL and we exemplified it with several queries on a real psychiatric database. All ST4SQL queries can be translated in equivalent SQL queries, thus the semantics of ST4SQL can be provided with respect to the SQL one.

As for the ongoing work, we want to explore issues related to the spatio-temporal querying (e.g., spatio-temporal index structures) in order to extend ST4SQL and optimize the translation of ST4SQL toward SQL and the evaluation of obtained spatio-temporal queries.

References

[1] F. Amaddeo and M. Tansella. Information systems for mental health. *Epidemiologia e psichiatria sociale*, 18:1–4, 2009.

[2] A. P. Association and A. P. A. T. F. on DSM-IV. *Diagnostic and statistical manual of mental disorders: DSM-IV-TR.* Amer Psychiatric Pub Inc, 2000.

[3] A. Belussi, C. Combi, and G. Pozzani. Formal and conceptual modeling of spatio-temporal granularities. In *Proceedings of the International Database Engineering and Applications Symposium, IDEAS 2009*, pages 275–283, Cetraro, Calabria, Italy, Sept. 2009. ACM.

[4] C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang. A glossary of time granularity concepts. *LNCS*, 1399:406–413, 1998.

[5] E. Camossi, M. Bertolotto, E. Bertino, and G. Guerrini. A multigranular spatiotemporal data model. In *GIS-03*, pages 94–101. ACM Press, Nov. 7–8 2003.

[6] C. X. Chen and C. Zaniolo. SQLST: A spatio-temporal data model and query language. In *ER*, pages 96–111, 2000.

[7] C. Combi, A. Montanari, and G. Pozzi. The T4SQL temporal query language. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007*, pages 193–202, Lisbon, Portugal, 6-10 2007. ACM.

[8] M. Erwig and M. Schneider. Developments in spatio-temporal query languages. In *DEXA Workshop*, pages 441–449, 1999.

[9] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM T. Database Syst.*, 25(1):1–42, 2000.

[10] IBM. DB2. URL: http://www.ibm.com/software/data/db2/.

[11] International Organization for Standardization. ISO/IEC 9075-14:2008 Information technology - Database languages - SQL - Part 14: XML-Related Specifications (SQL/XML). URL: http://www.iso.org.

[12] International Organization for Standardization. *ISO/IEC 13249-3:2006: Information technology — Database languages — SQL Multimedia and Application Packages — Part 3: Spatial.* 2006.

[13] C. S. Jensen, C. E. Dyreson, M. H. Böhlen, and et al. The consensus glossary of temporal database concepts - february 1998 version. In *Temporal Databases, Dagstuhl*, pages 367–405, 1998.

[14] V. Khatri, S. Ram, R. T. Snodgrass, and G. M. O'Brien. Supporting user-defined granularities in a spatiotemporal conceptual model. *Ann. Math. Artif. Intell*, 36(1-2), 2002.

[15] M. Koubarakis, T. K. Sellis, A. U. Frank, S. Grumbach, R. H. Güting, C. S. Jensen, N. A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, volume 2520 of *Lecture Notes in Computer Science*. Springer, 2003.

[16] Oracle Corporation. Oracle database. URL: http://www.oracle.com.

[17] N. L. Sarda. Hsql: A historical query language. In *Temporal Databases*, pages 110–140. 1993.

[18] A. P. Sistla and O. Wolfson. Temporal triggers in active databases. *IEEE Trans. Knowl. Data Eng.*, 7(3):471–486, 1995.

[19] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 422–432. IEEE Computer Society, 1997.

[20] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

[21] J. R. R. Viqueira and N. A. Lorentzos. SQL extension for spatio-temporal data. *VLDB J.*, 16(2):179–200, 2007.

[22] M. F. Worboys. Imprecision in finite resolution spatial data. *GeoInformatica*, 2(3):257–279, 1998.

[23] World Health Organization (WHO). International classification of diseases (ICD). URL: http://www.who.int/classifications/icd/en/.