The VLDB Journal manuscript No.
(will be inserted by the editor)

Jose R. Rios Viqueira · Nikos A. Lorentzos

SQL Extension for Spatio–Temporal Data

Received: date / Accepted: date

Abstract An SQL extension is formalised for the management of spatio-temporal data, i.e. of spatial data that evolves with respect to time. The extension is dedicated to applications such as topography, cartography and cadastral systems, hence it considers discrete changes both in space and in *time*. It is based on the rigid formalization of data types and of SQL constructs. Data types are defined in terms of time and *spatial quanta*. The SQL constructs are defined in terms of a kernel of *few* relational algebra operations, composed of the well–known operations of the 1NF model and of two more, *Unfold* and *Fold*. In conjunction with previous work, it enables the uniform management of 1NF structures that may contain not only spatio–temporal but also either purely temporal or purely spatial or conventional data. The syntax and semantics of the extension is fully consistent with the SQL:1999 standard.

Keywords Spatial Databases · Data Modelling · Spatio–temporal Databases · SQL

The final publication is available at Springer via http://dx.doi.org/10.1007/s00778-005-0161-9

Jose R. Rios Viqueira
Systems Laboratory, Department of Electronics and Computer Science, University of Santiago de Compostela, Instituto de Investigaciones Tecnolgicas, Campus Sur, 15782 Santiago de Compostela, A Coruña, Spain
Tel.: +34 981 520829
Fax: +34 981 520829
E-mail: joserios@usc.es

Nikos A. Lorentzos
Informatics Laboratory, Agricultural University of Athens, Iera Odos 75, GR 11855 Athens, Greece
Tel.: +30 210 529 4175
Fax: +30 210 529 4199
E-mail: lorentzos@aua.gr

1 Introduction

During the last two decades a lot of research has been undertaken in the areas of temporal [2,9,10,44, 55] and spatial [5,14,20,22,24,26,27,37,38,42,47,50–52, 54,56,58,59] data management. Most of temporal data models were dedicated to the management of valid time data, i.e. of data associated with the time during which these data are true in the real world. In their majority, these models attempted to embed the temporal functionality in the core of a Database Management System (DBMS) by the definition of new data types and operations.

The first generation spatial approaches, termed *GIS–centric* [12], provided data structures and operations for the direct manipulation of *maps* [3,16,27,28,56]. Relevant implementations attempted to link graphic objects with conventional data stored in separate files, missing, this way, the advantages of database technology. The more recent *GIS–centric* tools are implemented either on top of only a conventional DBMS (*layered architecture*) or on top of two different subsystems (*dual architecture*), (i) a conventional DBMS and (ii) a spatial subsystem. Clearly, neither of these two architectures takes full advantage of database technology, since conventional DBMSs do not provide efficient manipulation of spatial data. To overcome this limitation, a new research effort was next undertaken in the area of *spatial databases* [5,14,20,22,24,26,37,38,42, 47,50–52,54,58,59], leading to a new set of approaches, termed *DBMS–centric* in [12]. As a result of the research in this area, the last generation commercial and open–source DBMSs [29,30,46,1] include extensions of models that enable the storage and management of spatial data and a spatial SQL as well, compliant with already existing standards [35,45].

Finally, the combination of research in temporal and in spatial data management, gave rise to a new research area, *spatio–temporal databases*, which is concerned with the management of the changes of spatial data with respect to time [4,6,7,13,17–19,21,23,25,36,43,53,57, 62,64,66,67].

Despite however the large number of approaches for temporal, spatial and spatio–temporal data management, the authors of this paper argue that a completely satisfactory solution has not still been achieved, in the sense that specific limitations can be identified in every approach. In particular, a major problem is that operations on 2–d spatial objects have much individuality. It is noticed, for example, that three spatial types should in principle be defined in a spatial data model, *point*, *line* and *surface*, whose definition should match the common interpretation by humans. However, it is also noticed that the *spatial intersection* of two spatial objects of a surface type is, in the general case, a set consisting of (i) points, (ii) lines, (iii) surfaces and (iv) spatial objects composed of surfaces that are connected with lines. The situation becomes even more complex if time is also involved. To cope with this problem, the broad majority of approaches had to either (i) simplify the obtained results by losing pieces of spatial data or (ii) consider complex data types and complex data structures. As an example of the former, only the surface parts are obtained by the spatial intersection of two surfaces in [3, 16,20,24,27,29,30,49,52,54,59] whereas a second operation is required to obtain the line parts in [17,19,25, 26,64]. Regarding the latter, data types of the form *set of spatial objects* are adopted in [3,7,17,19,20,22,25,26, 28,37,38,46,51,64,66], spatio–temporal data types are considered in [7,17,19,23,25,43,66] and complex data structures, such as nested–relational, object–relational and object–oriented, are used in [5,47,49,51,58,59,67]. Also, object–relational structures are used in [29,30,46] to implement complex data types.

Additional limitations identified in the various spatial and spatio–temporal approaches can be resumed as follows. Some approaches restrict to only informal presentations either of the data types or of the functionality of the operations [4,5,7,14,24,27,36,38,43, 50,52–54,56,64,66,67]. The empty set is treated as a valid spatial object in [14,17,16,19,20,25,26,29,30,35, 37,38,45,51,52,64,66]. Non–closed valid spatial objects are considered in [51,58]. A relation may have at most one spatial attribute in [3,6,27,36,57]. Two types of attributes, *explicit* and *implicit*, have to be considered in [4]. Many–sorted algebras are defined in [24,54]. Various granularities of time are not supported in [4, 7,13,17–19,21,25,31,53,66,67]. Finally, the functionality of conventional algebra operations had to be redefined in [7,67].

In this paper, an SQL extension is formalised for the management of spatio–temporal data that overcomes the above limitations. It is dedicated to applications such as topography, cartography and cadastral systems, hence it considers discrete changes both in space and in time. Data types are defined in terms of *time*and *spatial* quanta. The SQL constructs are defined in terms of a kernel of *few* relational algebra operations, composed of the well–known operations of the 1NF model and of two more, *Unfold* and *Fold*. The present work extends further IXSQL [40], defined for the management of temporal data. In conjunction with [40], it enables the uniform management of conventional, temporal, spatial, and spatio–temporal data. It also achieves and generalizes further the functionality of other approaches. The spatial and spatio–temporal functionality can be implemented in the core of a 1NF relational DBMS, which can next be extended, in a generic manner, to a more powerful either nested–relational or object–relational model.

The remainder sections of this paper are outlined as follows. Sections 2 and 3 are devoted to the definition of quanta and data types for time and space, respectively. Predicates and functions, only those required in subsequent formalism, are defined in Section 4. Data structures and relational algebra operations are defined in Section 5. The syntax and semantics of the SQL extension are defined in Section 6. Some discussion on continues changes is made in Section 7. Section 8 summarises the characteristics of the approach, discusses further capabilities, reviews previous work and addresses implementation issues. Conclusions and issues of further research are drawn in the last section. The full syntax of the SQL extension is given in an Appendix.

2 Quanta and Data Types for Time

Definition 1 If N is the set of natural numbers and $n \in N, n > 0$, then $I_n = \{i \mid i \in N, 0 \leq i < n\}$ is called a *discrete 1-dimensional (1-d) space.* The elements of I_n are called *(1-d) points.*

Note that, by definition, I_n is a nonempty, finite, totally ordered set.

Definition 2 If $p, q \in I_n, p \leq q$, a *period* [p, q] over I_n is defined as the set

$$[p, q] \equiv \{i \mid i \in I_n \land p \leq i \leq q\}.$$

For the objectives of the present paper, it is assumed that I_n matches some *time* data type like those supported in SQL, i.e. DATE, TIME, TIMESTAMP. To avoid however restricting to a particular time data type, successive time points of a *generic type* are denoted as t_i, t_{i+1}, t_{i+2} etc, and periods of time instants are denoted as $[t_i, t_j]$. Based on this, the following definition is given.

Definition 3 Two generic types for time are defined:
– INSTANT $\equiv \{t_0, t_1, \ldots, t_{n-1}\}$.
– PERIOD $\equiv \{[t_i, t_j] \mid t_i, t_j \in \text{INSTANT} \land t_i \leq t_j\}$.

The elements of the first set are called (*time*) *instants* or *time quanta* and those of the second (*time*) *periods.* Figure 1 depicts the time instants of the generic data type INSTANT and periods over it.

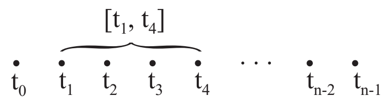


Fig. 1 Quanta for Time.

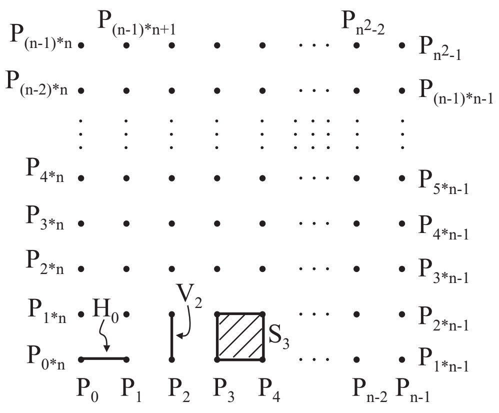


Fig. 2 Quanta for Space.

A period of particular interest is used in subsequent formalism $t_{\mathrm{TIME\_ALL}}$, defined as $t_{\mathrm{TIME\_ALL}} \equiv [t_0, t_{n-1}]$. By definition, INSTANT is totally ordered. Taking this into account, the following definition is given.

Definition 4 A set S of time instants is *connected* iff it matches the period $[\mathit{min}(t_i \in S), \mathit{max}(t_i \in S)]$.

3 Spatial Quanta and Spatial Data Types

In this section a series of successive definitions is provided that ends up with a rigid formalism of the spatial data types that are considered in this approach.

Definition 5 A *discrete 2-dimensional (2-d) space* is defined as a set $I_n \times I_n$, where I_n is a (1-d) space.

Based on this definition, the following types of spatial quanta are formalized.

Definition 6
Let $(i, j) \in I_n \times I_n$ and let $k = n * j + i \in I_{n^2}$. Then the set

- $P_k = \{(i, j)\}$ is called a *2-dimensional (2-d) spatial point* or a *2-d quantum point* or simply a *point*,
- $H_k \equiv \{(x, y) \in R^2 \mid i \le x \le i + 1 \wedge y = j\}$ is called a *pure horizontal quantum line* iff $0 \le i < n - 1$,
- $V_k \equiv \{(x, y) \in R^2 \mid x = i \wedge j \le y \le j + 1\}$ is called a *pure vertical quantum line* iff $0 \le j < n - 1$,
- $S_k \equiv \{(x, y) \in R^2 \mid i \le x \le i + 1 \wedge j \le y \le j + 1\}$ is called a *pure quantum surface* iff $0 \le i < n - 1$ and $0 \le j < n - 1$.

By definition, a point is a set of just one element of R^2 whereas a pure quantum line and a pure quantum surface is an infinite subset of R^2. A point (pure quantum line, pure quantum surface) is geometrically represented as a dot (line segment, square) on a plane. Figure 2 depicts the geometric representation of the following quanta: *points* $P_0, P_1, \ldots, P_{n^2-1}$, *pure horizontal quantum line* H_0, *pure vertical quantum line* V_2 and *pure quantum surface* S_3.

Clearly, the number of quanta defined this way is finite. As will be seen (Definition 10), an element of some *spatial data type* will be defined in terms of this finite set.

Definition 7 Let the set of all quantum points (pure quantum lines, pure quantum surfaces) be denoted as $Q_{\mathrm{POINT}}(Q_{\mathrm{PL}}, Q_{\mathrm{PS}})$. Then the following sets are also defined.

- $Q_{\mathrm{LINE}} \equiv Q_{\mathrm{PL}} \cup Q_{\mathrm{POINT}}$, called the *set of all quantum lines.*
- $Q_{\mathrm{SURFACE}} \equiv Q_{\mathrm{PS}} \cup Q_{\mathrm{LINE}}$, called the *set of all quantum surfaces.*

Based on the concept of spatial quanta, a series of data types for space are defined. Before the formalism is given, preliminary definitions for *quantum set* and *spatial connectivity* are provided.

Definition 8 If $\varnothing \neq S = q_1 \cup q_2 \cup \ldots \cup q_n \subset R^2$, where $q_i \in Q_{\mathrm{SURFACE}}$ $\forall i = 1, 2, \ldots, n$, S is called a *quantum set.*

Definition 9 A *quantum set* $S \subset R^2$ is called *connected* iff for every pair of elements $x, y \in S$ there exists a sequence of spatial quanta $q_1, q_2, \ldots, q_n \subseteq S$ that satisfies the following two properties:

1. $x \in q_1$ and $y \in q_n$.
2. $q_i \cap q_{i+1} \neq \varnothing$ for $i = 1, 2, \ldots, n - 1$.

Based on this definition, spatial data types are formalized.

Definition 10 Let g be a non-empty, connected quantum set. It is then defined that g is of a (*2-d spatial*) type

– POINT $\Leftrightarrow g \equiv q_i, \quad q_i \in Q_{\mathrm{POINT}}$.
– PLINE $\Leftrightarrow g \equiv \cup_i q_i, q_i \in Q_{\mathrm{PL}}$.
– LINE $\Leftrightarrow g \equiv \cup_i q_i, q_i \in Q_{\mathrm{LINE}}$.
– PSURFACE $\Leftrightarrow g \equiv \cup_i q_i, q_i \in Q_{\mathrm{PS}}$.
– SURFACE $\Leftrightarrow g \equiv \cup_i q_i, q_i \in Q_{\mathrm{SURFACE}}$.

An element of one of the above types is called, respectively, *(2-d spatial) point, pure line, line, pure surface* and *surface.*

Figure 3(a) depicts the geometric representation of the following spatial objects:

- *Points*: $P_0, P_1, \ldots, P_{168}$.
- *Pure lines*: (i), (ii), (iv) and (v). Object (iv) is the union of four pure quantum lines.
- *Lines*: Any of the previous pure lines and points.
- *Pure surfaces*: (iii), (vi) and (vii). Object (vi) is the union of eleven pure quantum surfaces. Object (vii) is a surface with a *hole.*

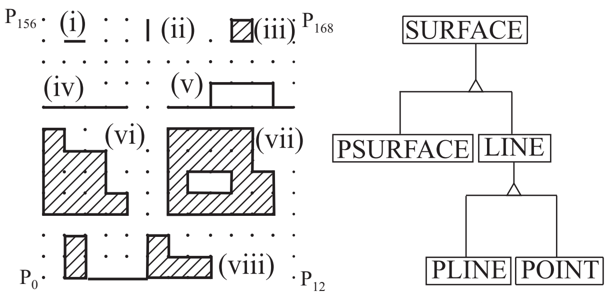


Fig. 3 Spatial Objects and Spatial Data Types.

– *Surfaces*: Any of the above pure surfaces, any of the above lines and object (viii), which is a *hybrid* surface.

Note that, by definition, a point is a unary *set*, it is *not* an element. Due to the fact that all other data types have also been defined as sets, it is possible to define set operations (union, intersection, etc.) involving a point and a spatial object of some other data type, as will be seen in the next section. As will also be seen, such set operations will also be possible to be applied between spatial objects of any two distinct types. Due to this, it is said that all the spatial objects are *spatially compatible*.

A spatial object of type PLINE is also of type SURFACE. Alternatively therefore, and in order to distinguish it from a *pure surface*, such an object is also called a *degenerate pure surface*. For a similar reason, an object of type POINT is also called a *degenerate pure line* and a *degenerate pure surface*.

Figure 3(b) illustrates the relationship between all the spatial data types. Two spatial objects of particular interest are used in subsequent formalism, g_{SURF_ALL} and g_{LINE_OUT}, defined as follows (Figure 2):

– $g_{SURF\_ALL} = S_0S_1S_2 \ldots S_{(n-1)*n-2} = \cup_i S_i$, where $S_i \in Q_{PS}$.
– $g_{LINE\_OUT} = L_{BOTTOM} \cup L_{RIGHT} \cup L_{TOP} \cup L_{LEFT}$, where

$$\begin{aligned} L_{BOTTOM} &= H_0 \cup H_1 \cup \ldots \cup H_{n-2}, \\ L_{TOP} &= H_{(n-1)*n} \cup H_{(n-1)*n+1} \cup \ldots \cup H_{n^2-2}, \\ L_{LEFT} &= V_0 \cup V_n \cup \ldots \cup V_{(n-2)*n}, \\ L_{RIGHT} &= V_{n-1} \cup V_{2*n-1} \cup \ldots \cup V_{(n-1)*n-1}. \end{aligned}$$

Hence, g_{SURF_ALL} is a pure surface decomposed of the union of all possible quanta. Also, g_{LINE_OUT} is a line composed of the four sides of a rectangle with corners the points P_0, P_{n-1}, P_{n^2-1} and $P_{(n-1)*n}$.

4 Predicates and Functions

Providing definitions for a *full* set of predicates and functions is beyond the objectives of this paper. Hence, the definitions below restrict only to those that are either necessary for the subsequent formalism or are used in examples.

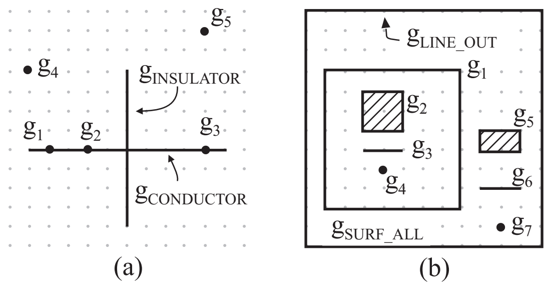


Fig. 4 Illustration of *conductive* and *surrounds*.

If $g_1, g_2, g_{CONDUCTOR}$ and $g_{INSULATOR}$ are spatial objects of any type, the next predicate enables determining whether there is a *path* from g_1 to g_2 in $g_{CONDUCTOR}$, which does not cross $g_{INSULATOR}$.

conductive $(g_A, g_B, g_{CONDUCTOR}, g_{INSULATOR}) \Leftrightarrow$
there exists a sequence of quanta $q_1, q_2, \ldots q_n \subseteq g_{CONDUCTOR}$ which satisfies the following three conditions:
1. $q_1 \subseteq g_A$.
2. $q_n \subseteq g_B$.
3. $(\forall i, 1 \le i \le n-1)$
$(\varnothing \neq q_i \cap q_{i+1} \nsubseteq g_{INSULATOR})$.

Considering the objects in Figure 4(a), it follows that *conductive*$(g_1, g_B, g_{CONDUCTOR}, g_{INSULATOR})$ evaluates to true if $g_B = g_2$. If g_B is any of g_3, g_4, g_5 the predicate evaluates to false. Let g_{LINE_OUT} and g_{SURF_ALL} be the spatial objects defined in Section 3 then, for the objects in Figure 4(b), *conductive*$(g_i, g_{LINE\_OUT}, g_{SURFACE\_ALL}, g_1)$ evaluates to true for i = 5, 6, 7, but it evaluates to false for i = 2, 3, 4.

The next predicate is defined in terms of *conductive* and is of spatial interest, as is witnessed by its name.

g_1 *surrounds* $g_2 \Leftrightarrow$
$\neg$*conductive*$(g_2, g_{LINE\_OUT}, g_{SURF\_ALL}, g_1)$.

Hence, for the objects in Figure 4(b), g_1 *surrounds* g_i, evaluates to true for i = 2, 3, 4, but it evaluates to false for i = 5, 6, 7. Note that, as opposed to *conductive*, g_1 *surrounds* $g_2 \neq g_2$ *surrounds* g_1.

If P_{k1}, P_{k2} are points with coordinates (i_{k1}, j_{k1}), (i_{k2}, j_{k2}), respectively, their Euclidian distance is defined as the real number

$$distance(P_{k1}, P_{k2}) = \sqrt{(i_{k2} - i_{k1})^2 + (j_{k2} - j_{k1})^2}.$$

Let g_A and g_B be two spatial objects of any data type. It is then defined that

$$distance(g_A, g_B) = min(distance(P_{A_i}, P_{B_j})),$$

where $P_{A_i}, P_{B_j} \in Q_{POINT}$, and $P_{A_i} \subseteq g_A$ and $P_{B_j} \subseteq g_B$.

R

Name	G
Marathon lake	g_1
Aoos river	g_2
Crystal spring	g_3

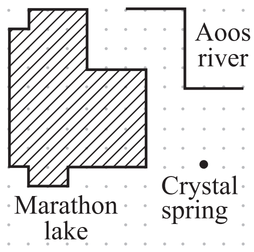


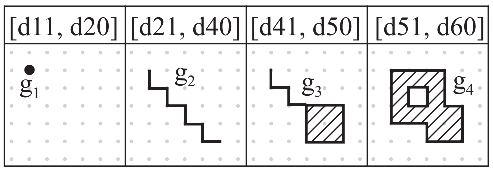
Fig. 5 Spatial Relation and its Geometric Representation.

5 Relational Algebra

A relation is defined in the known way, except that the underlying domain of one or more of its attributes can be of some time or space data type. In the sequel, R(**A**, **B**, **C**) denotes a relation, where **A**, **B**, **C** are three non-empty disjoint sets of one or more attributes whose data types are immaterial. T and G are used to denote, respectively, an attribute of a time or space data type. Moreover, TG (**TG**) denotes an attribute (set of attributes) of either a time or of a space type, exclusively. Finally, (**a**, **b**, **c**) denotes a tuple of a relation with scheme R(**A**, **B**, **C**), where **a** (**b**, **c**) denotes the values for attributes **A** (**B**, **C**). Similarly, t, g and tg, possibly subscripted, are used to denote, respectively, time instants, spatial objects and any of the two.

An example of a *spatial relation* is given in Figure 5. The geometric representation of the objects recorded in attribute R.G are a point, a pure line and a pure surface. Figure 6(a) gives an example of a *spatio-temporal relation* recording the evolution of a spatial object, Morpheas, with respect to time. Indeed, from the geometric representation of the values recorded in attribute Shape, it can be seen that during [d11, d20] Morpheas was a spring. Next, during [d21, d40], [d41, d50], and [d51, d60] it became, respectively, a river, a river flowing into a small lake and a big lake with an island. Note that Figure 6(b) shows another spatio-temporal relation, but its difference from that in Figure 6(a) is discussed later in this section. Two more examples of spatio-temporal relations are depicted in Figure 7. Specifically, LAND_USE records the evolution of land use with respect to time. The plots of these lands are given in the same figure. Note that the lifespan of object g3 (last tuple of LAND_USE) is [d31, d50]. However, two plots of this object are given, one during [d31, d40] and another for [d41, d50]. This discipline is followed in the sequel for other objects too, for ease of presentation, as will be realized later. Similarly, P_OWNER is used to record the owner and the shape of various land parcels during various periods of time.

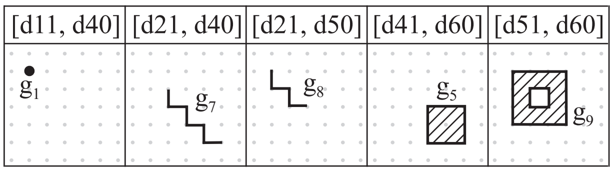
If $TG_1, TG_2, \ldots, TG_n$ is a set of attributes, where the data type of each TG_i is either SURFACE or PERIOD, then one relation of particular interest is $QUANTA\_ALL(TG_1, TG_2, \ldots, TG_n)$, which contains a single tuple $t = (tg_{ALL1}, tg_{ALL2}, \ldots, tg_{ALLn})$, where



H

Name	DG	DT	Shape	Time
Morpheas	1	5	g_1	[d11, d20]
Morpheas	1	5	g_2	[d21, d40]
Morpheas	1	5	g_3	[d41, d50]
Morpheas	1	5	g_4	[d51, d60]

(a) Spatio-temporal Relation Normalised on Space and Time



H_1

Name	DG	DT	Shape	Time
Morpheas	1	5	g_1	[d11, d40]
Morpheas	1	5	g_7	[d21, d40]
Morpheas	1	5	g_8	[d21, d50]
Morpheas	1	5	g_5	[d41, d60]
Morpheas	1	5	g_9	[d51, d60]

(b) Non-normalised Spatio-temporal Relation

Fig. 6 Normalised and Non-normalised Spatio-temporal Relations.

tg_{ALLi} is g_{SURF_ALL}, if the data type of TG_i is SURFACE, and it is t_{TIME_ALL} otherwise.

Based on the previous data structures, relational operations are also defined. In particular, the model incorporates the well-known relational operations, *Union*, *Except*, *Project*, *Cartesian Product*, *Select*, *Intersect*, *Join* etc. For the remainder operations, it is preliminarily defined that *set*(x) ≡ x if x is not a set and *set*(x) ≡ x otherwise. Based on this, some more operations are defined next.

Definition 11 If R is a relation with scheme R(**A**, TG) then relation

$$U = \mathit{Unfold}[TG](R)$$

has scheme U(**A**, TG), where the data type of U.TG is that in Figure 8 and contents

$$\{(a, q_i) \mid q_i \text{ is a quantum} \wedge \mathit{set}(q_i) \subseteq \mathit{set}(tg) \wedge (a, tg) \in R\}.$$

For an example of the application of *Unfold* on a time attribute, consider relation H_1 in Figure 6(b). Then the result of $UH_1 = \mathit{Unfold}[Time](H_1)$ is shown in Figure 9(a). For an example of the application of this operation on a spatial attribute, let R be

LAND_USE

Use	Shape	Time
Cultivation	g_1	[d11, d20]
Forest	g_2	[d11, d30]
Forest	g_4	[d31, d50]
Industrial	g_1	[d21, d30]
Industrial	g_3	[d31, d50]

P_OWNER

Owner	Pid	Shape	Time
John	P2	g_5	[d21, d40]
Peter	P1	g_6	[d21, d30]
Peter	P1	g_8	[d31, d40]
Peter	P3	g_7	[d41, d60]
Susan	P3	g_7	[d31, d40]
Susan	P1	g_8	[d41, d60]
Susan	P2	g_5	[d41, d60]

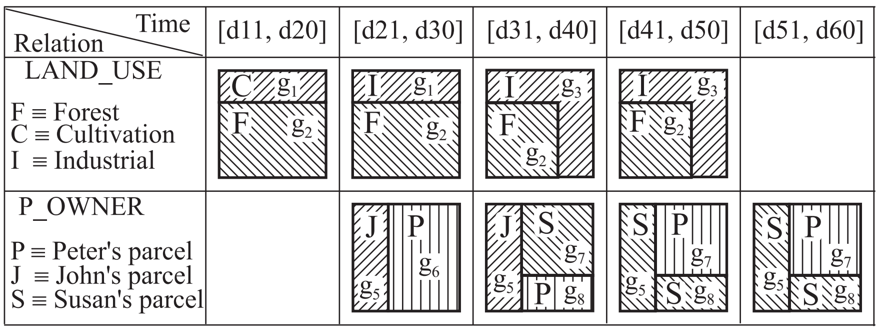


Fig. 7 Spatio-temporal Relations.

	R.TG	*Unfold*[TG](R)	*Fold*[TG](R)
TIME	INSTANT	INSTANT	PERIOD
	PERIOD	INSTANT	PERIOD
SPACE	POINT	POINT	POINT
	PLINE	LINE	PLINE
	LINE	LINE	LINE
	PSURFACE	SURFACE	PSURFACE
	SURFACE	SURFACE	SURFACE

Fig. 8 Result Data Types in *Unfold* and *Fold*.

the relation in Figure 5. Then U = *Unfold*[G](R) is depicted in Figure 10(a). Such result consists of pure quantum surfaces (Figure 10(b)), pure quantum lines (Figure 10(c)) and points (Figure 10(d)).

Definition 12 If R is a relation with scheme R(**A**, TG) then relation

F = *Fold*[TG](R)

has scheme F(**A**, TG), where the data type of F.TG is that in Figure 8, and contents

$\{(a, tg = \bigcup_{i=1}^{n} \mathit{set}(tg_i)) \mid$
$(tg \text{ is } \mathit{connected}) \wedge ((a, tg_i) \in R, i = 1, 2, \ldots, n) \wedge$
$(\nexists (a, tg_{n+1}) \in R \text{ such that}$
$\mathit{set}(tg) \cup \mathrm{set}(tg_{n+1}) \text{ is } \mathit{connected})\}.$

For an example of the application of *Fold* on a time attribute, if UH_1 is the relation in Figure 9(a), then *Fold*[Time](UH_1) yields relation H_1 in Figure 6(b). To illustrate its application on a spatial attribute, consider a relation $R(\mathbf{A}, G) = \{(a, g_{Ai})\} \cup \{(a, g_{Bi})\}$, where g_{Ai} and g_{Bi} are the spatial objects depicted in Figure 11. Then, the result of F = *Fold*[G](R) yields a relation $F(\mathbf{A}, G) = \{(g_{Ui})\}$, where the geometric representation of each g_{Ui} is shown in Figure 12(a).

UH_1

Name	DG	DT	Shape	Time
Morpheas	1	5	g_1	d11
...	...	...	...	...
Morpheas	1	5	g_1	d40
Morpheas	1	5	g_7	d21
...	...	...	...	...
Morpheas	1	5	g_7	d40
Morpheas	1	5	g_8	d21
...	...	...	...	...
Morpheas	1	5	g_8	d50
Morpheas	1	5	g_5	d41
...	...	...	...	...
Morpheas	1	5	g_5	d60
Morpheas	1	5	g_9	d51
...	...	...	...	...
Morpheas	1	5	g_9	d60

(a) *Unfold*[Time](H_1)

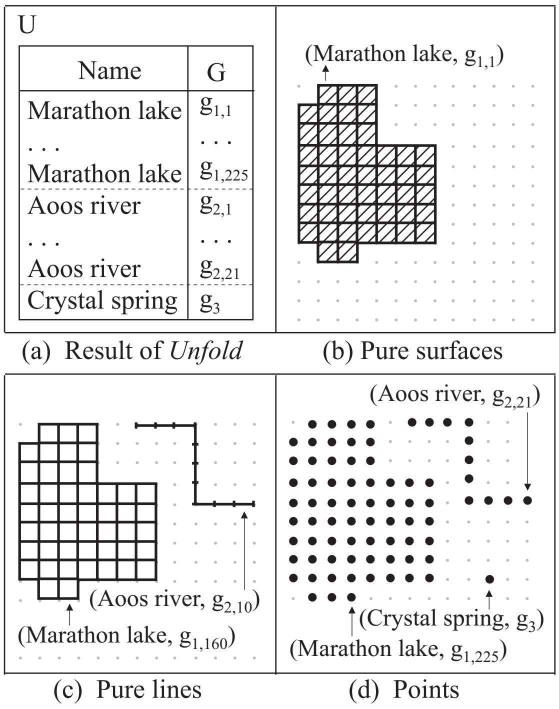
UH_2

Name	DG	DT	Shape	Time
...	...	...	...	...
Morpheas	1	5	$q_{8,1}$	d50
Morpheas	1	5	$q_{8,2}$	d50
Morpheas	1	5	$q_{8,3}$	d50
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Morpheas	1	5	$q_{8,n}$	d50
...	...	...	...	...

Each $q_{i,j}$ denotes one of the quanta in g_i

(b) *Unfold*[Shape](UH_1)

Fig. 9 Illustration of operation *Unfold*.


U

Name	G
Marathon lake	$g_{1,1}$
. . .	. . .
Marathon lake	$g_{1,225}$
Aoos river	$g_{2,1}$
. . .	. . .
Aoos river	$g_{2,21}$
Crystal spring	g_3

Fig. 10 Illustration of operation *Unfold* on spatial data.

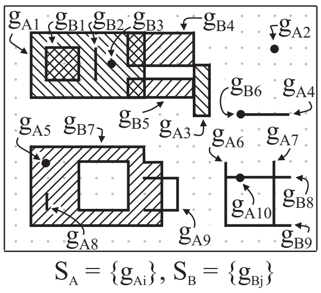

g_{A1} g_{B1} g_{B2} g_{B3} g_{B4} g_{A2}
g_{B6} g_{A4}
g_{A5} g_{B7} g_{B5} g_{A3} g_{A6} g_{A7}
g_{B8} g_{A10}
g_{A8} g_{A9} g_{B9}
$S_A = \{g_{Ai}\}$, $S_B = \{g_{Bj}\}$

Fig. 11 Input Spatial Data.

Operations *Unfold* and *Fold* can be generalised so as to be applied to various time and space attributes as follows.

Definition 13 Let $R(A, TG_1, TG_2, \ldots, TG_n)$ be a relation, then it is defined that:

$$\begin{aligned}&\mathit{Unfold}[TG_1, TG_2, \ldots, TG_n](R) \equiv \\ &\quad \mathit{Unfold}[TG_n](\ldots(\mathit{Unfold}[TG_2](\mathit{Unfold}[TG_1](R)))). \\ &\mathit{Fold}[TG_1, TG_2, \ldots, TG_n](R) \equiv \\ &\quad \mathit{Fold}[TG_n](\ldots(\mathit{Fold}[TG_2](\mathit{Fold}[TG_1](R)))).\end{aligned}$$

Although in [40] it has been argued that these operations can be applied to a relation on some attribute of any data type, for the objectives of this paper only space and time attributes are considered. To illustrate their functionality, consider relation H_1 in Figure 6(b). It is a spatio-temporal relation and, by a careful examination, it can be seen that it contains duplicate data. As an example, consider its fourth tuple and the geometric representation of object g_5. It is then noted that during the period [d51, d60] (which is a sub-period of the lifespan of g_5), object g_5 contains the top left quantum surface. From the geometric representation of object g_9, it is noted that during [d51, d60] this quantum surface is also contained in object g_9. Hence, a piece of surface has been recorded redundantly in two distinct tuples during [d51, d60]. A similar observation applies to objects g_7 and g_8 for the period [d21, d40]. This data duplication can be eliminated by the application of $UH_2 = \mathit{Unfold}[\text{Shape}, \text{Time}](H_1)$, where UH_2 is shown in Figure 9(b). Since UH_2 does not contain duplicate data, then the same is also true for relation H (Figure 6(a)) resulting from $H = \mathit{Fold}[\text{Shape}, \text{Time}](UH_2)$. It is also noticed that the application of *Fold* first on a space and then on a time attribute yields a relation that enables identifying the evolution of the shape of spatial data with respect to time.

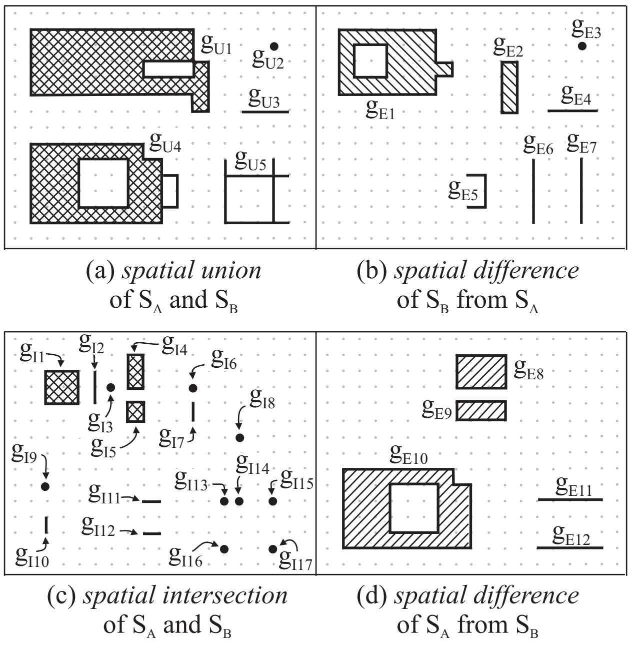

g_{U1} g_{U2} g_{U3} g_{U4} g_{U5}
(a) *spatial union* of S_A and S_B
g_{E1} g_{E2} g_{E3} g_{E4} g_{E5} g_{E6} g_{E7}
(b) *spatial difference* of S_B from S_A
g_{I1} g_{I2} g_{I3} g_{I4} g_{I5} g_{I6} g_{I7} g_{I8} g_{I9} g_{I10} g_{I11} g_{I12} g_{I13} g_{I14} g_{I15} g_{I16} g_{I17}
(c) *spatial intersection* of S_A and S_B
g_{E8} g_{E9} g_{E10} g_{E11} g_{E12}
(d) *spatial difference* of S_A from S_B

Fig. 12 Result of Spatial Operations.

To complete this formalism, one more operation *Normalise* is introduced, that simplifies the formulation of queries.

Definition 14 Let $R(\mathbf{A}, \mathbf{TG})$ be a relation, then it is defined that:

$$\mathit{Normalise}[\mathbf{TG}](R) \equiv \mathit{Fold}[\mathbf{TG}](\mathit{Unfold}[\mathbf{TG}](R))$$

Thus, relation H in Figure 6(a) can be obtained directly from relation H_1 in Figure 6(b) with the expression: $H = \mathit{Normalise}[\text{Shape}, \text{Time}](H_1)$.

6 SQL Extension

Based on the relational algebra of the previous section, a spatio-temporal SQL extension is formalized. The functionality of the extension is defined in terms of primitive SQL expressions. Note that in [61] this functionality has been defined in terms of relational algebra operations. Keywords are given in **bold**. The full SQL syntax is given in the Appendix.

6.1 Query Specification

Two new optional clauses, <reformat clause> and <normalise clause>, have been added to the syntax of the SQL:1999 query specification, which is thus extended as follows [40] (full syntax details are given in the Appendix):

SELECT	[<set quantifier>] <select list>	(1)
FROM	<table ref list>	(2)
[**WHERE**	<search condition>]	(3)
[**GROUP BY**	<grouping column ref list>]	(4)
[**HAVING**	<search condition>]	(5)
[<reformat clause>]		(6)
[<normalise clause>]		(7)
[**ORDER BY**	<sort spec list>]	(8)

The BNF syntax of the new constructs is as follows:

```
<reformat clause> ::=
   REFORMAT AS <reformat item>

<reformat item> ::=
   FOLD [ALL] <reformat column list>
      [<reformat item>]
   | UNFOLD [ALL] <reformat column list>
      [<reformat item>]

<normalise clause> ::=
   NORMALISE ON[ALL]
      <reformat column list>

<reformat column list> ::=
   <reformat column>
      [{, <reformat column>}...]

<reformat column> ::=
   <column reference> | <unsigned integer>
```

Rule 1 The <reformat column list> must be a sub–list of the attributes that appear in <select list>.

(Note that in SQL:1999 line (8) is not actually part of <query specification>, but it has been included here for simplicity reasons. Note also that, as is known, literals may be included at various places in the previous syntax. The definition of time and space literals can be found in [40] and [61], respectively. Full syntax details are also given in the Appendix below). The semantics and functionality of this extension is described below.

6.1.1 Reformat Clause

Lines (1)-(5) are executed as in SQL:1999 and next lines (6)-(8) are executed in this order. The <reformat clause> enables the introduction of a sequence of *Unfold* and *Fold* operations that are applied to the result produced by the execution of lines (1)-(5). Formally, let $\mathrm{TR}_0(\mathbf{A}, \mathbf{TG_1}, \mathbf{TG_2}, \ldots, \mathbf{TG_m})$ be the scheme of the relation obtained by execution of lines (1)-(5). Let also **XFOLD** denote either **UNFOLD** or **FOLD**. Then, the result obtained by

REFORMAT AS
  XFOLD $\mathbf{TG_1}$, **XFOLD** $\mathbf{TG_2}$, …,
  XFOLD $\mathbf{TG_m}$

is a relation $\mathrm{TR_m}$ that matches the result of the sequence of m relational algebra operations

$\mathrm{TR_i} = \mathit{XFold}[\mathbf{TG_i}](\mathrm{TR_{i-1}})$, i = 1, 2, …, m.

Hence, an example of such a valid clause is:

REFORMAT AS
  UNFOLD $\mathrm{T_1}$, $\mathrm{G_1}$, $\mathrm{G_2}$,
  FOLD $\mathrm{T_2}$, $\mathrm{G_3}$, $\mathrm{TG_2}$

6.1.2 Normalise Clause

The <normalise clause> enables the application of a *Normalise* operation to the relation obtained by the execution of lines (1)-(6).

Formally, if $\mathrm{TR}(\mathbf{A}, \mathrm{TG_1}, \mathrm{TG_2}, \ldots, \mathrm{TG_n})$ is the relation obtained by the execution of lines (1)-(6), then the SQL expression

NORMALISE ON $\mathrm{TG_1}, \mathrm{TG_2}, \ldots, \mathrm{TG_n}$

is equivalent to the relational algebra expression

$\mathit{Normalise}[\mathrm{TG_1}, \mathrm{TG_2}, \ldots, \mathrm{TG_n}](\mathrm{TR})$.

Since duplicate tuples are allowed in SQL, a variation of **UNFOLD**, **FOLD** and **NORMALISE** is also provided that enables obtaining relations with duplicate tuples. This functionality is achieved by the incorporation of the [**ALL**] option. Such option is also incorporated in the remainder of SQL extension but, its discussion is beyond the objectives of this paper.

6.1.3 Sort Rows on the Basis of Space and Time Columns

The total ordering defined for spatial objects in [61] and time periods in [40] enables incorporating references to attributes of these types after the key words ORDER BY. In addition, aggregate functions can be applied to attributes of these types.

6.2 Query Expression

In SQL:1999, binary operations are involved in a <query expression>. Such an expression can be either a <non–join query expression> or a <joined table> and is extended further in Subsection 6.3 and 6.6, respectively. Moreover, one more expression is added, <unary query expression>, described in Subsection 6.7. Hence, the syntax of a <query expression> becomes

```
<query expression> ::=
    <IXSQL non–join query expression>
  | <IXSQL joined table>
  | <IXSQL unary query expression>
```

All of these extensions make use of the following algorithm, originally presented in [39], which enables the easy formulation of queries that return data which evolves with respect to time. This algorithm is described below.

Let R(**A**), S(**B**) be two non-temporal relations and let

R OPERATION S

denote any SQL:1999 binary operation. Let also TR(**A**, T), TS(**B**, T) be the respective temporal relations, i.e., they record the contents of R and S, respectively, for various time instants. It is then defined that

TR OPERATION [ALL] **EXPANDING**(T) TS

returns a relation with scheme and contents deduced by the execution of the following five steps:

S1 Let UR = TR UNFOLD [ALL] (T),
US = TS UNFOLD [ALL] (T).

S2 Let TIME be the relation returned by the expression

```
SELECT T FROM UR
UNION
SELECT T FROM US
```

S3 For every $t \in$ TIME, let UR_t and US_t be the relations returned, respectively, by the expression

```
SELECT A FROM UR WHERE T = 't'
SELECT B FROM US WHERE T = 't'
```

S4 For every $t \in$ TIME, let $UP_t(C)$ be the scheme of the relation obtained by the SQL:1999 binary operation

UR_t OPERATION [ALL] US_t

S5 It is then defined that

TR OPERATION [ALL] **EXPANDING**(T) TS

returns a relation P(**C**, T), where the domain of T is of a period type. The rows of P are those obtained by steps S5.1 and S5.2 below.

S5.1 For every $t \in$ TIME,
if c is a row in UP_t
then add a row (c, t) in UP(**C**, T).

S5.2 P = UP **NORMALIZE** [**ALL**] (T).

Definition 15 Given '*OPERATION*' as above, it is said that '*OPERATION **EXPANDING(T)**' is its evolution with respect to time.*

6.3 Non–Join Query Expression

The syntax of the SQL:1999 non–join query expression, has been extended to support two sets of binary operations, namely quantum and pair–wise operations.

6.4 Quantum Operations

In a simplified case (full syntax details are given in the Appendix), the syntax, to incorporate within SQL the quantum operations, is the following:

```
<non–join query expression> ::=
    <query exp 1> UNION [ALL]
        [EXPANDING (<reformat column list>)]
        <query exp 2>
  | <query exp 1> EXCEPT [ALL]
        [EXPANDING (<reformat column list>)]
        <query exp 2>
  | <query exp 1> INTERSECT [ALL]
        [EXPANDING (<reformat column list>)]
        <query exp 2>
```

Rule 2 <query exp 1> and <query exp 2> must return union compatible relations.

Rule 3 <reformat column list> must form a sub–list of the attributes of the relations returned by both <query exp 1> and <query exp 2>.

The functionality achieved by the inclusion of the **EXPANDING** option has already been described in the previous section for time attributes. Notice, that spatial attributes can also be included in the <reformat column list>, achieving this way spatial and spatio–temporal functionality. Such functionality is next illustrated by examples.

To start with spatial data, let $R1(\mathbf{A}, G) = \{(\mathbf{a}, g_{Ai})\}$, $R2(\mathbf{A}, G) = \{(\mathbf{a}, g_{Bj})\}$ be two relations, where the geometric representation of each g_{Ai} and each g_{Bj} is depicted in Figure 11. Then, the expression

```
SELECT * FROM R1          (1)
UNION EXPANDING (G)       (2)
SELECT * FROM R2          (3)
```

yields a relation with scheme (**A**, G) and contents $\{(\mathbf{a}, g_{Uk})\}$, where the geometric representation of each g_{UK} is depicted in Figure 12(a). Similarly, if line (2) in the expression above is replaced by "**EXCEPT EXPANDING** (G)" ("**INTERSECT**

I

Shape	Time
g_1	[d21, d30]
g_3	[d31, d50]

S

Shape	Time
g_7	[d31, d40]
g_9	[d41, d60]

QU

Shape	Time
g_1	[d21, d30]
g_{17}	[d31, d40]
g_{18}	[d41, d50]
g_9	[d51, d60]

QE

Shape	Time
g_1	[d21, d30]
g_{11}	[d31, d40]
g_{19}	[d31, d40]
g_{20}	[d41, d50]

QI

Shape	Time
g_{20}	[d31, d40]
g_{11}	[d41, d50]
g_{19}	[d41, d50]

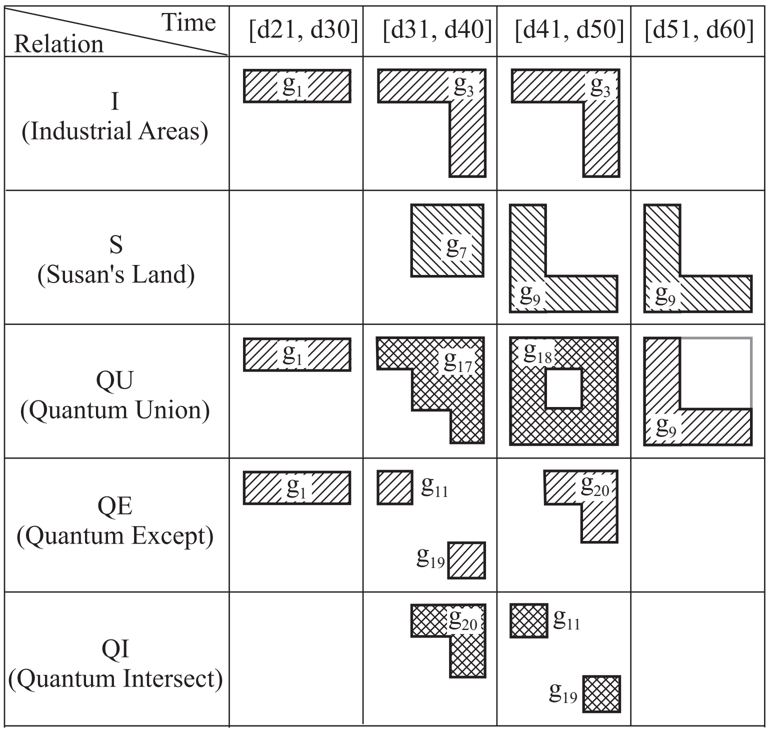


Fig. 13 Illustration of Quantum Operations on Spatio–temporal Data.

EXPANDING (G)"), then the contents of the result relation are $\{(\mathbf{a}, g_{\mathrm{Ek}})\}$ ($\{(\mathbf{a}, g_{\mathrm{IK}})\}$), where the geometric representation of each g_{EK} (g_{IK}) is depicted in Figure 12(b) (Figure 12(c)).

To illustrate the application of the extension to spatio–temporal data consider, initially, relations LAND_USE and P_OWNER in Figure 7. Then, relations I (evolution with respect to time of industrial areas) and S (evolution with respect to time of Susan's land), both in Figure 13, are obtained, respectively, by the two following expressions:

```
SELECT Shape, Time        SELECT Shape, Time
FROM   LAND_USE           FROM   P_OWNER
WHERE                     WHERE Owner = 'Susan'
   Use = 'Industrial'     NORMALISE ON
                                    Shape,Time
```

Then the expression

```
SELECT * FROM I                        (1)
UNION EXPANDING (Shape, Time)          (2)
SELECT * FROM S                        (3)
```

yields a relation whose scheme and contents match those of relation QU in Figure 13. Similarly, if line (2) in the expression above is replaced by "**EXCEPT EXPANDING** (Shape, Time)" ("**INTERSECT EXPANDING** (Shape, Time)"), then the scheme and contents of the result relation match those of relation QE (QI), also in Figure 13.

6.5 Pair–Wise Operations

In a simplified case (full syntax details are given in the Appendix), the syntax, to incorporate within SQL the pair–wise operations, is the following:

```
<non–join query expression>::=
   <query exp 1>
      WUNION [ALL] OF (<ref col list 1>)
      [EXPANDING (<ref col list 2>)]
      <query exp 2>
   | <query exp 1>
      WEXCEPT [ALL] OF (<ref col list 1>)
      [EXPANDING (<ref col list 2>)]
      <query exp 2>
   | <query exp 1>
      WINTERSECT [ALL] OF (<ref col list 1>)
      [EXPANDING (<ref col list 2>)]
      <query exp 2>
```

Rule 4 <ref col list 1> and <ref col list 2> must form sub–lists of the list of attributes of the relations returned by both <query exp 1> and <query exp 2>.

Rule 5 <ref col list 1> and <ref col list 2> may not have attributes in common.

Note that sets of columns returned by <query exp 1> and <query exp 2> do not have to be disjoint because SQL:1999 does not impose such a restriction. If R1($\mathbf{A}, \mathbf{TG}$), R2($\mathbf{B}, \mathbf{TG}$) are the relations returned by

<query exp 1> and <query exp 2>, respectively, then the result obtained by

<query exp 1> **WUNION OF** (TG) <query exp 2>

matches the one obtained by the expression

```
SELECT  R1.A, R2.B, R1.TG   (1)
FROM    R1, R2              (2)
UNION EXPANDING (TG)        (3)
SELECT  R1.A, R2.B, R2.TG   (4)
FROM    R1, R2              (5)
```

Similarly, if line (3) in the previous expression is replaced by either of

```
EXCEPT     EXPANDING (TG)   (3)
INTERSECT  EXPANDING (TG)   (3)
```

then the result of the expression matches, respectively, the one obtained by either of the two following expressions:

```
<query exp 1>
   WEXCEPT OF (TG) <query exp 2>

<query exp 1>
   WINTERSECT OF (TG) <query exp 2>
```

The functionality of the extension is next illustrated by examples. Regarding spatial data, consider relations R1(**A**, G) = $\{(\mathbf{a}, g_{Ai})\}$, R2(**B**, G) = $\{(\mathbf{b}, g_{Bj})\}$ where the geometric representation of each g_{Ai} and each g_{Bj} is depicted in Figure 11. Then, the expression

```
SELECT * FROM R1   (1)
WUNION OF (G)      (2)
SELECT * FROM R2   (3)
```

yields a relation with scheme (**A**, **B**, G) and contents $\{(\mathbf{a}, \mathbf{b}, g_{Uk})\}$, where the geometric representation of each g_{UK} is depicted in Figure 12(a). Similarly, if line (2) in the expression above is replaced by "**WEXCEPT OF** (G)" ("**WINTERSECT OF** (G)"), then the contents of the result relation are $\{(\mathbf{a}, \mathbf{b}, g_{Ek})\}$ ($\{(\mathbf{a}, \mathbf{b}, g_{IK})\}$), where the geometric representation of each g_{EK} (g_{IK}) is depicted in Figure 12(b) (Figure 12(c)).

For spatio–temporal data consider, again, relations LAND_USE and P_OWNER in Figure 7. Then, relations I (evolution with respect to time of industrial areas) and S (evolution with respect to time of land Owned by either Susan or Peter), both in Figure 14, are obtained, respectively, by the two following expressions:

```
SELECT Use, Shape,          SELECT Owner, Shape,
         Time                        Time
FROM   LAND_USE             FROM   P_OWNER
WHERE                       WHERE
   Use = 'Industrial'          Owner = 'Susan'
                            or Owner = 'Peter'
                            NORMALISE ON
                                     Shape,Time
```

Then the expression

```
SELECT * FROM I             (1)
WUNION OF (Shape, Time)     (2)
SELECT * FROM S             (3)
```

yields a relation whose scheme and contents match those of relation WU in Figure 14. Similarly, if line (2) in the expression above is replaced by "**WEXCEPT OF** (Shape, Time)" ("**WINTERSECT OF**(Shape, Time)"), then the scheme and contents of the result relation match those of relation WE (WI), also in Figure 14.

To conclude with pair–wise operations, note that the **EXPANDING** option can also be included. The functionality achieved by this keyword has already been described in Subsection 6.2.

6.6 Joined Table

Beyond the extension by the **EXPANDING** option of the various types of the SQL:1999 join operations [39], a syntax has also been provided for joined tables, that enables the explicit application of various types of overlay operations. This syntax is given below.

```
<IXSQL overlay> ::=
   <table ref 1> [NATURAL][<overlay type>]
      OVERLAY [ALL] [OF (<ref col list 1>)]
      [EXPANDING (<ref col list 2>)]
      <table ref 2>

<overlay type> ::=
   INNER
   | {LEFT | RIGHT | FULL} [OUTER]
```

Rule 6 Exactly one of the options, either **NATURAL** or **OF** (<ref col list 1>) must be specified.

If the option <overlay type> is not specified then **INNER** is assumed by default. The option **NATURAL** is equivalent to the option **OF** (<ref col list>), where <ref col list> is the list of all attributes in the result of both <query exp 1> and <query exp 2>. Recall that <ref col list 1> and <ref col list 2> must be disjoint sub–lists of the attributes of the relations returned by <query exp 1> and <query exp 2>. If R1(**A**, **TG**), R2(**B**, **TG**) are two relations then the result obtained by

R1 **INNER OVERLAY OF** (TG) R2

matches the one obtained by the expression

I

Use	Shape	Time
Industrial	g_1	[d21, d30]
Industrial	g_3	[d31, d50]

S

Owner	Shape	Time
Susan	g_7	[d31, d40]
Susan	g_9	[d41, d60]
Peter	g_6	[d21, d30]
Peter	g_8	[d31, d40]
Peter	g_7	[d41, d60]

WU

Use	Owner	Shape	Time
Industrial	Susan	g_1	[d21, d30]
Industrial	Susan	g_{17}	[d31, d40]
Industrial	Susan	g_{18}	[d41, d50]
Industrial	Susan	g_9	[d51, d60]
Industrial	Peter	g_{22}	[d21, d30]
Industrial	Peter	g_{23}	[d31, d40]
Industrial	Peter	g_{17}	[d41, d50]
Industrial	Peter	g_7	[d51, d60]

WE

Use	Owner	Shape	Time
Industrial	Susan	g_1	[d21, d30]
Industrial	Susan	g_{11}	[d31, d40]
Industrial	Susan	g_{19}	[d31, d40]
Industrial	Susan	g_{20}	[d41, d50]
Industrial	Peter	g_{11}	[d21, d30]
Industrial	Peter	g_{24}	[d31, d40]
Industrial	Peter	g_{11}	[d41, d50]
Industrial	Peter	g_{19}	[d41, d50]

WI

Use	Owner	Shape	Time
Industrial	Susan	g_{20}	[d31, d40]
Industrial	Susan	g_{11}	[d41, d50]
Industrial	Susan	g_{19}	[d41, d50]
Industrial	Peter	g_{25}	[d21, d30]
Industrial	Peter	g_{19}	[d31, d40]
Industrial	Peter	g_{20}	[d41, d50]

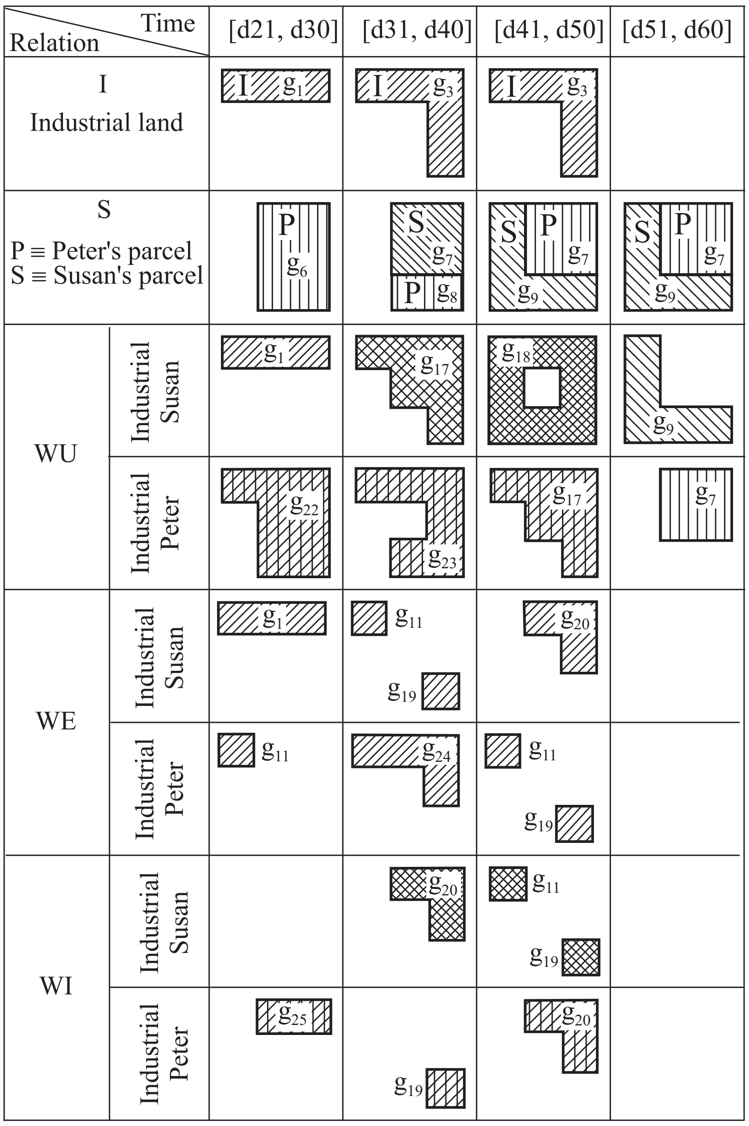


Fig. 14 Illustration of Pair–Wise Operations on Spatio–temporal Data.

```
SELECT *                                 (1)
FROM                                     (2)
    (SELECT *                            (3)
    FROM    R1                           (4)
    REFORMAT AS UNFOLD TG)               (5)
  INNER JOIN                             (6)
    (SELECT *                            (7)
    FROM    R2                           (8)
    REFORMAT AS UNFOLD TG)               (9)
  USING (TG)                            (10)
REFORMAT AS FOLD TG                     (11)
```

Similarly, if line (6) in the previous expression is replaced by either of

```
LEFT JOIN    (6)
RIGHT JOIN   (6)
FULL JOIN    (6)
```

then the result of the expression matches, respectively, the one obtained by either of

```
R1 LEFT  OVERLAY OF (TG) R2
R1 RIGHT OVERLAY OF (TG) R2
R1 FULL  OVERLAY OF (TG) R2
```

The spatial and spatio–temporal functionality of overlay operations is next illustrated by examples. For spatial data, consider relations R1(**A**, G) = $\{(\mathbf{a}, g_{Ai})\}$, R2(**B**, G) = $\{(\mathbf{b}, g_{Bj})\}$ where the geometric representation of each g_{Ai} and each g_{Bj} is depicted in Figure 11. Consider also relations I(**A**, **B**, G)={(**a**, **b**, g_{IK})} (g_{IK} in Figure 12(c)), L(**A**, **B**, G)={(**a**, **b**, g_{EK})} (g_{EK} in Figure 12(b)) and R(**A**, **B**, G)={(**a**, b, g_{EK})} (g_{EK} in Figure 12(d)). Then, the expression

```
SELECT * FROM R1          (1)
INNER OVERLAY OF (G)      (2)
SELECT * FROM R2          (3)
```

yields a relation whose scheme and contents match those of relation I. Similarly, if line (2) in the expression above is replaced by "**LEFT OVERLAY OF** (G)" ("**RIGHT OVERLAY OF** (G)", "**FULL OVERLAY OF** (G)"), then the scheme and contents of the result relation match those of the relation $I \cup L$ ($I \cup R$, $I \cup L \cup R$).

For spatio–temporal data consider, again, relations LAND_USE and P_OWNER in Figure 7. Then, relations I (evolution with respect to time of industrial areas) and S (evolution with respect to time of Susan's land), both in Figure 15, are obtained, respectively, by the two following expressions:

```
SELECT Use, Shape,        SELECT Owner, Shape,
       Time                      Time
FROM   LAND_USE           FROM   P_OWNER
WHERE                     WHERE  Owner = 'Susan'
   Use = 'Industrial'     NORMALISE ON
                                 Shape,Time
```

Then the expression

```
SELECT * FROM I                     (1)
FULL OVERLAY OF (Shape, Time)       (2)
SELECT * FROM S                     (3)
```

yields a relation whose scheme and contents match those of relation FO in Figure 15. The subsets of relation FO achieved by the other types of overlay operations are also shown in the same Figure 15.

Finally, note that the **EXPANDING** option can also be included. The functionality achieved by this keyword has already been described in Subsection 6.2.

6.7 Unary Query Expression

The definition of <unary query expression> enables incorporating in SQL the functionality of the unary operations *Complementation*, *Boundary*, *Envelope* and *Buffer*. An obvious revision of the algorithm given in Definition 15 is also considered, which enables defining the *evolution of these operations with respect to time*. Hence, the syntax is as follows:

```
<unary query expression> ::=
    (<unary query expression>)
    |<table ref>{COMPLEMENTATION
                | BOUNDARY
                | ENVELOPE [ALL]}
        OF (<ref col list 1>)
        [EXPANDING (<ref col list 2>)]
    | <table ref> BUFFER [ALL]
        OF (<ref col list 1>)
        WITHIN DISTANCE (<ref col list 3>)
        [EXPANDING (<ref col list 2>)]
```

Rule 7 <ref col list 1>, <ref col list 2> and <ref col list 3> must form disjoint sub–lists of the attributes of the relations returned by <table ref>.

Rule 8 The data type of attributes in <ref col list 3> must be numeric.

Rule 9 The number of attributes in <ref col list 2> must be identical to the number of attributes in <ref col list 3>.

Let R be a relation with scheme R(**A**, **TG**) and QUANTA_ALL(**TG**) be the relation defined in Section 5. Then the result obtained by

```
R COMPLEMENTATION OF (TG)
```

matches the one obtained by the following expression:

```
SELECT * FROM QUANTA_ALL
WEXCEPT OF (TG)
SELECT * FROM R
```

Similarly, the result given by

```
R BOUNDARY OF (TG)
```

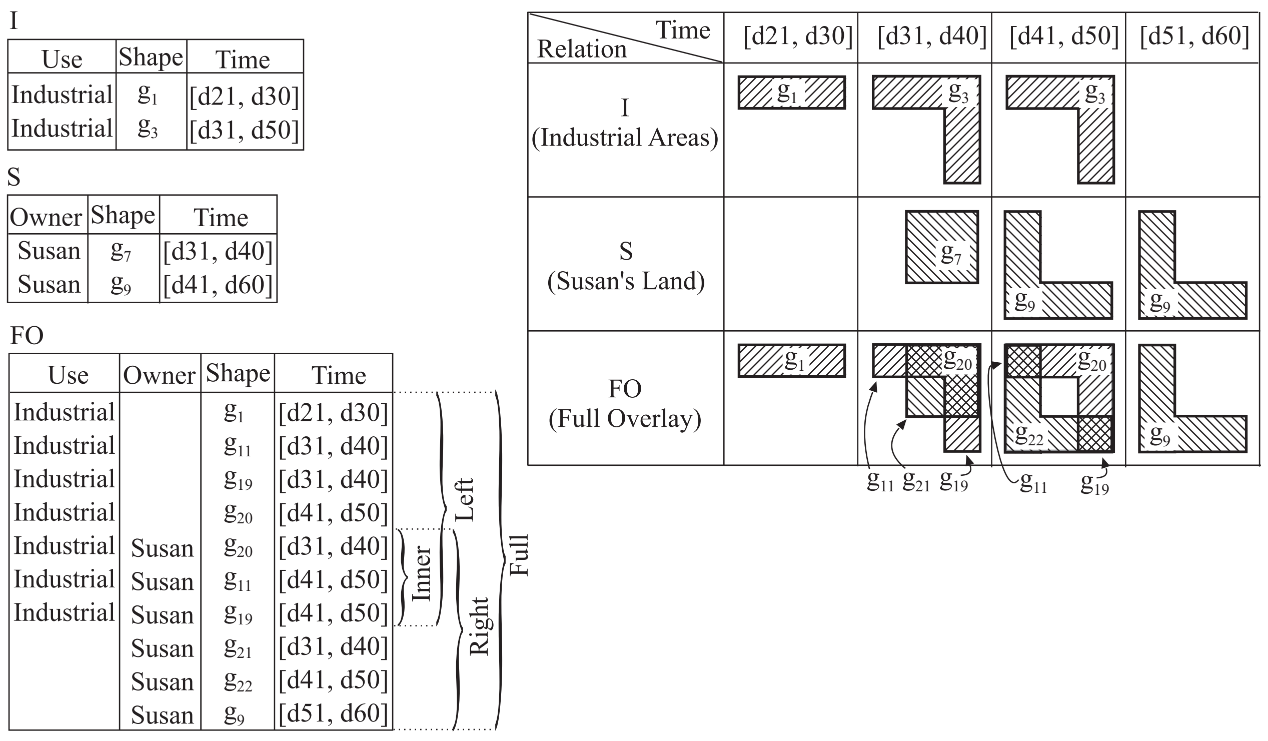
matches the one obtained by the following expression:

```
R COMPLEMENTATION OF (TG)
INTERSECT EXPANDING (TG)
SELECT * FROM R
```

Finally, the result given by

```
R ENVELOPE OF (TG)
```

matches the one obtained by the following expression:



I

Use	Shape	Time
Industrial	g_1	[d21, d30]
Industrial	g_3	[d31, d50]

S

Owner	Shape	Time
Susan	g_7	[d31, d40]
Susan	g_9	[d41, d60]

FO

Use	Owner	Shape	Time
Industrial		g_1	[d21, d30]
Industrial		g_{11}	[d31, d40]
Industrial		g_{19}	[d31, d40]
Industrial		g_{20}	[d41, d50]
Industrial	Susan	g_{20}	[d31, d40]
Industrial	Susan	g_{11}	[d41, d50]
Industrial	Susan	g_{19}	[d41, d50]
	Susan	g_{21}	[d31, d40]
	Susan	g_{22}	[d41, d50]
	Susan	g_9	[d51, d60]

Fig. 15 Illustration of Overlay Operations on Spatio–temporal Data.

```
SELECT NR.A, UQA.TG                              (1)
FROM (SELECT *                                   (2)
    FROM    R                                    (3)
    NORMALISE ON TG) AS NR,                      (4)
    (SELECT *                                    (5)
    FROM    QUANTA_ALL                           (6)
    REFORMAT AS UNFOLD TG) AS UQA                (7)
WHERE NR.TG surrounds UQA.TG                     (8)
REFORMAT AS FOLD TG                              (9)
```

Notice that if **TG** is a set of attributes {TG1, TG2, ..., TGn} then the expression in line (8) above has to be replaced by the expression

NR.TG1 *surrounds* UQA.TG1 and
NR.TG2 *surrounds* UQA.TG2 and . . . and
NR.TGn *surrounds* UQA.TGn

Let R be a relation with scheme R(**A**, **TG**, **D**), where the attributes in **D** are of some numeric data type, then the result obtained by

R **BUFFER OF**(TG) **WITHIN DISTANCE** (D)

matches the one obtained by the following expression:

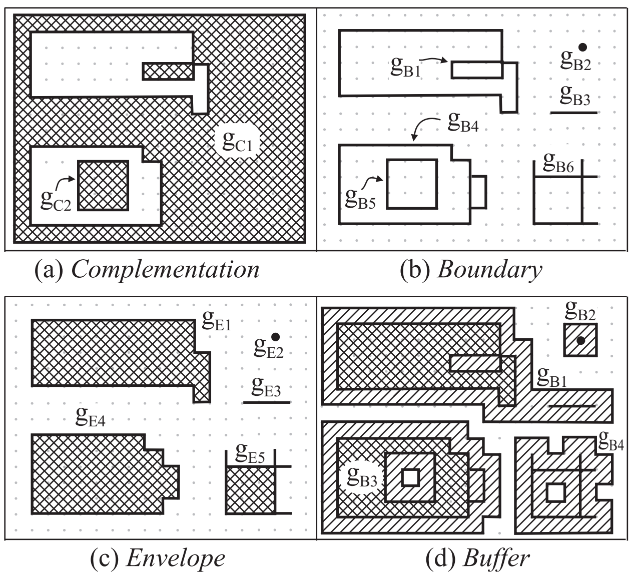
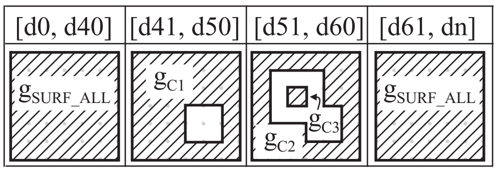


Fig. 16 Results of Unary Spatial Operations.

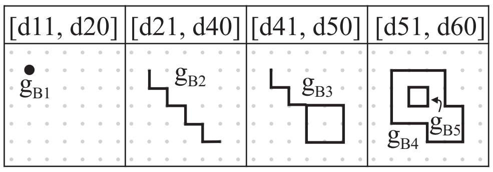
```
SELECT R.A, UQA.TG, R.D                          (1)
FROM    R,                                       (2)
    (SELECT *                                    (3)
    FROM QUANTA_ALL                              (4)
    REFORMAT AS UNFOLD TG) AS UQA                (5)
WHERE distance(R.TG, UQA.TG) < R.D               (6)
REFORMAT AS FOLD TG                              (7)
```



C

Name	DG	DT	Shape	Time
Morpheas	1	5	g_{SURF_ALL}	[d0 , d40]
Morpheas	1	5	g_{C1}	[d41, d50]
Morpheas	1	5	g_{C2}	[d51, d60]
Morpheas	1	5	g_{C3}	[d51, d60]
Morpheas	1	5	g_{SURF_ALL}	[d61, dn]

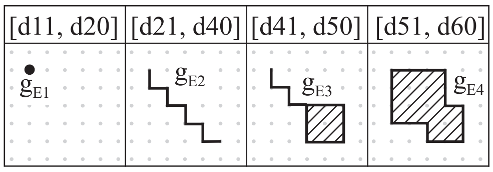
(a) *Complementation*



BO

Name	DG	DT	Shape	Time
Morpheas	1	5	g_{B1}	[d11, d20]
Morpheas	1	5	g_{B2}	[d21, d40]
Morpheas	1	5	g_{B3}	[d41, d50]
Morpheas	1	5	g_{B4}	[d51, d60]
Morpheas	1	5	g_{B5}	[d51, d60]

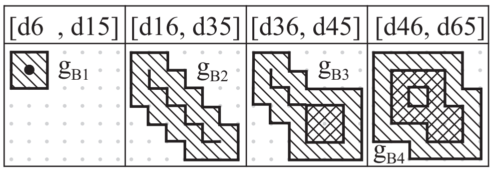
(b) *Boundary*



E

Name	DG	DT	Shape	Time
Morpheas	1	5	g_{E1}	[d11, d20]
Morpheas	1	5	g_{E2}	[d21, d40]
Morpheas	1	5	g_{E3}	[d41, d50]
Morpheas	1	5	g_{E4}	[d51, d60]

(c) *Envelope*



BU

Name	DG	DT	Shape	Time
Morpheas	1	5	g_{B1}	[d6 , d15]
Morpheas	1	5	g_{B2}	[d16, d35]
Morpheas	1	5	g_{B3}	[d36, d45]
Morpheas	1	5	g_{B4}	[d46, d65]

(d) *Buffer*

Fig. 17 Illustration of Unary Operations on Spatio–temporal Data.

Notice that if **TG**, **D** are two sets of attributes, respectively, TG1, TG2, ..., TGn and D1, D2, ..., Dn, then the expression in line (6) above has to be replaced by the expression

distance(R.TG1, UQA.TG1) < R.D1 and
distance(R.TG2, UQA.TG2) < R.D2 and ... and
distance(R.TGn, UQA.TGn) < R.Dn

The spatial and spatio–temporal functionality of these operations is next illustrated by examples. For spatial data, consider a relation R(**A**, D, G) = (**a**, 1, g_{Ui}), where the geometric representation of each g_{Uk} is depicted in Figure 12(a). Then, each of the expressions

– R **COMPLEMENTATION OF** (G)
– R **BOUNDARY OF** (G),
– R **ENVELOPE OF** (G),
– R **BUFFER OF**(G) **WITHIN DISTANCE** (D),

give a relation with scheme (**A**, D, G) and with respective contents

– {(**a**, 1, g_{Ck})} (g_{Ck} in Figure 16(a)),
– {(**a**, 1, g_{Bk})} (g_{Bk} in Figure 16(b)),
– {(**a**, 1, g_{Ek})} (g_{Ek} in Figure 16(c)),
– {(**a**, 1, g_{Bk})} (g_{Bk} in Figure 16(d)).

Regarding spatio–temporal data, if H is the relation in Figure 6(a), then the expressions

– H **COMPLEMENTATION OF** (Shape, Time)
– H **BOUNDARY OF** (Shape, Time),
– H **ENVELOPE OF** (Shape, Time),
– H **BUFFER OF**(Shape, Time)
  WITHIN DISTANCE (DG, DT),

give, respectively, a relation whose scheme and contents match those of relation

– C in Figure 17(a),
– BO in Figure 17(b),
– E in Figure 17(c),
– BU in Figure 17(d).

7 Continues Changes in Time

Although it has explicitly been stated that the proposed model is dedicated to the management of discrete changes in time, an outline of some application to continues changes is presented in this section. Consider therefore the relation in Figure 18(a), used to record flights. The interpretation, e.g. for of the first two tuples, FLIGHTS

Fid	Time	Position
A	[t10, t19]	P_1
A	[t20, t29]	P_2
A	[t40, t49]	P_3
A	[t50, t79]	P_4
A	[t80, t89]	P_5
B	[t20, t29]	P_6
B	[t30, t39]	P_7
B	[t40, t49]	P_8
B	[t50, t59]	P_9

(a)

R

Fid	B_T	E_T	B_P	E_P
A	t10	t20	P_1	P_2
A	t40	t50	P_3	P_4
A	t50	t80	P_4	P_5
B	t20	t30	P_6	P_7
B	t30	t40	P_7	P_8
B	t40	t50	P_8	P_9

(b)

Fig. 18 Illustration of continuous changes in time.

is that flight A was at position p_1 and p_2 at time t_{10} and t_{20}, respectively. Then the relation in Figure 18(b) can be obtained by the IXSQL [40] expression

```
SELECT Fid,
       start(F1.Time) AS B_T,
       start(F2.Time) AS E_T,
       F1.Position    AS B_P,
       F2.Position    AS E_P
FROM   FLIGHT AS F1, FLIGHT AS F2
WHERE  F1.Fid = F2.Fid AND
       succ(end(F1.Time), 1) = start(F2.Time)
```

The first row shows that at times t_{10} and t_{20} flight A was at positions p_1 and p_2, respectively. Assume also that a function *f* enables computing the position of a flight at some intermediate time instant. Then the query '*give the position of all flights at time t45*' can be formulated as

```
SELECT Fid,
       f(B_T, E_T, B_P, E_P) AS P
FROM   R
WHERE  B_T <= t45 AND t45 <= E_T
```

Function *f* can be selected from a list. Note however that the management of continuous changes in time cannot be exhausted by a simple example, i.e. further research is required for the application of the proposed model in this area. Approaches however to handle continues (and also discrete) changes are addressed in [17,19,25] (although none of them aims at defining a data model). Observations on them are the following.

Many data types are considered. The spatial types are POINT, POINTS (set of points), LINE (set of lines) and REGION (set of surfaces). A variety of spatio-temporal data types are also defined. One example is MAPPING(A), where A is one of the above spatial data types. If i_k represents a time interval and f_k represents a function from the time instants in i_k to A then a value of type MAPPING(A) is a set of the form $\{(i_1, f_1), (i_2, f_2), \ldots, (i_n, f_n)\}$. Hence, [17,19,25] enable defining many functions f_k and each of them is defined at the time of data recording (as opposed to the example given earlier, where this definition was provided at the time of query formulation). At the same time, however, the modification of a function, for experimentation purposes, requires the prior update of the contents of the database.

Note that three types of continues changes in time can generally be identified, (i) the change of position of an object, discussed above, (ii) the change of its shape and (iii) the change of both its position and shape. However, it is not always easy to define an interpolation function for cases (ii) and (iii). This problem can become even more difficult if the interpolation has to consider spatial types of the form *set of spatial objects*. As a final observation, [17,19,25] do not define some spatial data type that consists of objects whose geometric representation consists of both points and lines and surfaces. (The authors of the present paper estimate that this is easy to fix.). As a consequence, however, it seems that currently continues changes like those in Figures 5 and 6 of this work cannot currently be modelled in [17, 19,25].

8 Discussion

The spatio–temporal SQL extension defined in the present paper has been designed to manipulate discrete changes both in space and in time. Regarding the representation of space, it is close to raster-based approaches, although it considers points, lines and surfaces. Regarding the management of time, it considers valid time recorded at the level of tuple. The characteristics of the proposed model can be outlined as follows:

Formalism: Although many models have been defined for the management of spatial and spatio-temporal data, many of them lack formalism [4,7,14,27,36,38,50,52,54, 56,64,66,67]. Contrary to this, a rigid formalism has been developed in the present paper for the definition of the data types (time and space), the relational algebra operations and the SQL constructs.

Data Types: A minimum set of data types is supported: Two generic types for time (*instant*, *period*), three types for space (*point*, *pure line* and *pure surface*) and two more, *line* (either a pure line or a point) and *surface* (either a pure surface or a line). The first five match those used by humans in daily practice. The last two enable the uniform management of every type of spatial data. As opposed to this, other models consider sets of time instants [7,22,23], sets of time intervals [17,19,25, 43], *sets of spatial objects* [3,7,17,19,20,22,25,26,28,37, 38,46,51,64,66]. Only surfaces are supported in [18,20] and points are not supported in [13]. Only one generic spatial data type is supported in [3,7,22,23,37,38,46, 51,56,58,66,67]. Only surfaces without holes or lines with only two end–points are supported in [24,29,30,35, 36,45,47,49,54,59,64]. Note that all the spatial objects are set–theoretically *closed.* Note also that since the model is dedicated to applications such as topography, cartography and cadastral systems, hence there was no need to define specific spatio–temporal data types mainly useful for the management of moving objects [7,17,19, 23,25,43,66].

The empty set is treated as a valid period in [17,19,21,25,66] and as a valid set of spatial objects in [14,17,16,19,20,25,26,29,30,35,37,38,45,51, 52,64,66]. It seems however that such a set does not have practical interest and indeed, the proposed model does not make use of this set.

Hybrid Surfaces, composed of pure surfaces connected by pure lines, are valid spatial objects. Such a spatial object can have practical interest. For example, a hydrological information system may necessitate recording, as a whole, lakes and rivers pouring into them. However, such objects are not supported in [13,17,16,19,20, 24–26,31,36,47,49,52,54,59].

Data Structures: The proposed model makes use of the simple 1NF data structures for the recording of spatio–temporal data. As opposed to this, nested structures are considered in [5,51,58], and some form of an *object–relational model* is adopted in [47,49,59]. A combination of two structures, *relations* and *layers* is considered in [3, 27,36], and, as a result, two distinct types of operations are defined, one to handle relations and another to handle layers. Moreover, it does not enforce limitations on the data structures. Hence, a relation may have more than one attribute of either a spatial or time type. As opposed to this, a data structure may have at most one spatial attribute in [3,27,36]. Two or more spatial attributes are allowed in the data structures of [16, 28] but only one of them may be involved in spatial operations. Finally, *explicit* and *implicit* attributes are used in [4].

Operations: In many approaches, limitations are enforced on the functionality of operations related to the management of spatial or spatio–temporal data. For example, *Overlay* is defined only between surfaces in [3,18, 16,24,26]. *Quantum Intersection* is supported only partially in [3,24,49,50,52,59,64,66], and none of the *Quantum Operations* is supported in [13,14,36,43,47]. Contrary to these, in the proposed model *Overlay* applies not only to surfaces but also to lines as well as to spatial objects of distinct geometric representations, such as between a surface and a line. Note that such a result has not only theoretical but also practical interest.

More generally, *all* the relational operations are applicable to *all* the types of data. In particular, it has been shown [40] that operations *Unfold* and *Fold* can also be applied to conventional data. It thus follows that all the operations can be applied *uniformly* to any type of data, i.e. conventional, temporal, spatial and spatio–temporal. It also follows, that the model is of general purpose.

From the previous discussion, it follows that all the relational operations are closed, in that they are applied to relations and they always return relations.

In many approaches, the individuality of spatial and spatio–temporal operations has also led to limitations in their definition. For example, the spatial intersection of two surfaces is defined in a way that discards the line parts in [3,18,16,20,24,27,29,30,49,52,54,59, 64] but this definitely results in data loss. To overcome this problem, two such operations are defined in [19,25], one to obtain only the surface and another to obtain only the line parts. Similarly, only points are obtained by the spatial intersection of two lines in [24,50,54,64]. Contrary to these approaches, in the proposed model the application of one operation suffices to obtain all the result spatial objects without any data loss.

In their majority, the *DBMS–centric approaches* maintain the operations of a data model and define functions for the management of spatial or spatio–temporal data. Contrary to this, one characteristic of the proposed model is that the management of spatial and spatio-temporal data actually reduces to the management of relations, in that the user achieves the manipulation of spatial data by the application of operations to relations (*injection* of operations on spatial objects into the relational model). Moreover, a *map* of spatial objects can be seen as the geometric representation of the spatial data recorded in one or more relations. Hence, the approach combines fully the flexibility of *GIS–Centric approaches* [3,16,27,28,56] with the advantages of database technology.

One advantage of the proposed model is that all the operations on relations are defined in terms of few kernel relational operations. This set consists of the known relational operations and two more, *Unfold* and *Fold.*

Functionality additional to that described in the present paper can be outlined as follows [61]: (i) obtaining the holes of a set of pure surfaces [5,64], (ii) obtaining the end–points of a set of pure lines [5,29,30, 35,45,47,54], (iii) *splitting* a set of spatial objects with respect to some others [5,54], (iv) obtaining the *nearest neighbours* of spatial objects [24,26,30,46,50] and (v) obtaining the *Voronoi diagram* of a set of objects [5,24, 54,59]. In fact, the relevant operations defined in [61] achieve a more general functionality than that in [5,24, 26,29,30,35,45–47,50,54,59,64]

Finally, initial investigation [63] has shown that, in conjunction with the recursive capabilities of SQL:1999, the approach can also be used for the formalization of operations related to the management of *continuous* changes in space. Note that such changes have much practical interest but relevant operations have been proposed only informally in [56].

It follows, therefore, that the model is of general purpose, in that it can be applied uniformly to any type of data such as temporal, interval, spatial and spatio–temporal.

Spatial Compatibility: As has been shown, it is possible to record in the same attribute spatial objects whose geometric representation can be either a *point* or a *pure line* or a *pure surface* (Figure 5). Due to this, it is always possible to record in a relation the temporal evolution of any spatial object with respect to time. This flexibility is not supported in [13,17,16,19,24–26,36,47,49,52,54,59]. Given in addition that all the operations can be applied to all spatial objects, it is thus said that all these objects are *spatially compatible.* Note incidentally that evolution of spatial operations is not supported in [31,36,43,53,57, 67].

SQL Constructs: The syntax and semantics of the extension is fully consistent with the SQL:1999 standard.

Dimension Independent Model: The majority of the approaches restrict to the management of 2–d data [3,5, 6,16,19,24–26,29,30,28,35,36,38, 45–47,49,50,52,54,66]. Only some of them handle both 2–d and 3–d data [14,59,64,67]. As opposed to this, the extension of the proposed model, so as to apply to n–d, n $\geq$ 3, spatial data is straightforward.

As a final remark, it is recalled that the approaches in [17,19,25] (Section 7) also support discrete changes. This is achieved by the use of constant functions, whose definition is clearly simple. Beyond however the observations on these approaches in the previous and the present section, the estimation of the authors of this paper is that the model proposed in the present paper is simpler because (i) it considers few and simple data types, those of daily practice and (ii) it does not necessitate the use of functions. Overall, the authors of this paper are not aware of any other approach that satisfies all the above properties.

Similarities with other research approaches are outlined as follows: Elements resembling the *spatial quanta* of the present work are used in [65], however, the objective of [65] is just the incorporation of the 9–intersection model [15] in raster–based approaches.

Raster cells or pixels, which resemble the *pure quantum surfaces* of the present work, are used in [20, 48]. Therefore, neither pure quantum lines nor points are supported in these approaches. Besides, pixels have been considered in [48] only for issues related to image processing. Similarities can also be identified between the spatial data types of the present SQL extension and those proposed in [5]. However important differences are the following: Contrary to the present work, the model in [5] makes use of set–valued attributes, all the operations are primitive, operations are described informally, hybrid surfaces are supported only partially, spatial difference is not defined and a vector–based representation is considered. Operations resembling *Unfold* and *Fold* are used in [20,58]. However, the functionality of these operations differs radically from that in the present paper. Specifically, *Unfold* returns an infinite set of tuples, therefore, the model is conceptual, as is also admitted in [58]. Besides, two types of a *Fold* operation are defined, which return elements of some *set of points* type. In [20], only two operations are defined. The first one, which resembles operation *Unfold*, decomposes each tuple of a relation into so many tuples as the number of *raster cells* that are contained in a spatial object. The second, which resembles operation *Fold*, merges tuples with matching values on a given conventional attribute.

The SQL version defined in the present paper is a further extension of the IXSQL defined in [40]. Note that IXSQL [40] is dedicated to the management of interval (and subsequently to temporal) data, i.e. it does not handle spatial data. Even however for the management of temporal data, IXSQL considers intervals of the form [x, y), as opposed to the [x, y] intervals in the present work, which represent a potential ISO standard [34,32, 33]. Finally, there is also a difference in the formalism, in that temporal quanta are not defined in IXSQL. Part of a relational algebra (*Unfold*, *Fold* and *Quantum operations*), dedicated solely to the management of spatial data, is presented in [42] but the formalism for *Unfold* and *Fold* is different than that in the present paper. In addition, only four spatial data types are considered in [42], whose formalism differs from that in the present paper. A limited part of the SQL extension (*Unfold*, *Fold*, *Quantum Operations* and *Full Overlay*) for the management of spatio–temporal data is also presented in [62]. However, a distinct definition for each of the operations *Unfold* and *Fold* is provided, depending on whether it applies to a relation on a time or space attribute. Also, the syntax for *Full Overlay* is both different than that in the present paper and also of limited functionality. Finally an informal description of the afore–mentioned operations is provided in [60].

An implementation of the temporal part of the present SQL extension has already been undertaken as part of previous work [40]. Such an implementation includes the application of optimization techniques that are based on a study of the properties of operations *Unfold* and *Fold* [41]. Besides, an internal *Split* operation, applied instead of *Unfold*, reduces drastically the execution time (see also [11]). Furthermore, none of the relevant operations that have been implemented [40] requires the explicit execution of *Unfold.*

Regarding the part related to the management of spatial and spatio–temporal data, our investigation has shown that the results obtained in temporal data management are directly applicable to the implementation of the spatial part. As an initial effort, it is noted that, for a prototype, it only suffices to implement operations *Unfold* and *Fold*, and predicate *conductive*, given that all the others are expressed in terms of them and the conventional operations (relevant pseudo code is provided in [61]). Besides, a *canonical form* [61], gives a unique internal representation for all spatial objects. It should however be noted that the formalism in this paper does not necessarily imply that only a raster–based implementation should be undertaken. Indeed, a tight coupling of the logical with the physical level (vector or raster–based implementation) contradicts data independence. Initial investigation has shown that an efficient implementation can be based on a vector–based approach. Relevant research in this area must consider, amongst others, that current commercial DBMSs do not provide direct support of all the spatial data types proposed in this model, therefore an implementation based on them must consider the development of an appropriate interface.

9 Conclusions

An SQL extension for the management of spatio–temporal data has been developed, whose definition has been based on a formal extension of the relational model. The authors of this paper estimate that its further extension to an object–relational model can enable the easy and efficient implementation of commonly used spatial standards [35,45]. Further research concerns the formalization of a complete set of predicates and functions, including predicates to test topological relationships [8,15], a deeper investigation on the application of the present SQL extension to the management of continuous changes in space and in time. Finally, an efficient implementation should consider storage structures, optimisation techniques and an appropriate mapping to vector-based approaches.

Acknowledgements The authors are thankful to the reviewers for their substantial contribution in the improvement of this work. This work has partially been supported by the European Union, TMR Project CHOROCHRONOS (FMRX-CT96-0056). Additionally, the work of Jose R.R. Viqueira has been partially funded by CICYT (refs. TIC2002-04413-C04-04 and TIC2003-06593), and Xunta de Galicia (ref. PGIDIT02SIN10501PR).

References

1. Postgis: Geographic objects for postgresql. Retrieved April 2005 from http://www.postgis.org/
2. Ariav, G.: A temporally oriented data model. ACM Transactions on Database Systems **11**(4), 499–527 (1986)
3. Bentley Systems Inc: MicroStation GeoGraphics Users Guide Version 7.2 (2001). Found at http://docs.bentley.com
4. Böhlen, M.H., Jensen, C.S., Skjellaug, B.: Spatio-temporal database support for legacy applications. In: Proceedings of the 1998 ACM symposium on Applied Computing (SAC'98), Atlanta, GA, February 27 – March 1, pp. 226–234. ACM (1998)
5. Chan, E.P.F., Zhu, R.: QL/G – A query language for geometric data bases. In: Proceedings of the 1st International Conference on GIS, Urban Regional and Environmental Planning, Samos, Greece, April 19-21, pp. 271–286 (1996)
6. Chen, C.X., Zaniolo, C.: SQLST: A spatio-temporal data model and query language. In: Proceedings of the 19th International Conference on Conceptual Modeling (ER-2000), Salt Lake City, Utah, October 9-12, pp. 96–111 (2000)
7. Cheng, T.S., Gadia, S.K.: A pattern matching language for spatio-temporal databases. In: Proceedings of the 3th International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, November 29 - December 2, pp. 288–295 (1994)
8. Clementini, E., Di Felice, P.: A model for representing topological relationships between complex geometric features in spatial databases. Information Sciences **90**(1–4), 121–136 (1996)
9. Clifford, J., Croker, A.: The historical relational data model (HRDM) revisited. In: A. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass (eds.) Temporal Databases: Theory, Design and Implementation, pp. 6–27. Benjamin/Cummings, Redwood City, CA (1993)
10. Clifford, J., Tansel, A.U.: On an algebra for historical relational databases: Two views. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, pp. 247–265. ACM SIGMOD Record 14(4) (1985)
11. Date, C.J., Darwen, H., Lorentzos, N.A.: Temporal Data and the Relational Model. Morgan Kaufmann Publishers, San Fransisco, California (2003)
12. Davis, J.R.: Ibm's db2 spatial extender: Managing geo-spatial information within the dbms. Tech. rep., IBM Corporation (1998)
13. d'Onofrio, A., Pourabbas, E.: Formalization of temporal thematic map contents. In: Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems (GIS 2001), Atlanta, GA, November 9-10, pp. 15–20 (2001)
14. Egenhofer, M.J.: Spatial sql: A query and presentation language. IEEE Transactions on Knowledge and Data Engineering **6**(1), 86–95 (1994)
15. Egenhofer, M.J., Herring, J.R.: Categorizing binary topological relations between regions, lines, and points in geographic databases. Tech. rep., Department of Surveying Engineering, University of Maine (1992)
16. Environmental Systems Research Institute, Inc.: ArcInfo 8: A New GIS for the New Millennium (2000). Found at http://www.esri.com
17. Erwig, M., Güting, R.H., Schneider, M., Vazirgiannis, M.: Spatio-temporal data types: An approach to modeling and querying moving objects in databases. GeoInformatica **3**(3), 269–296 (1999)
18. Erwig, M., Schneider, M.: The honeycomb model of spatio-temporal partitions. In: Proceedings of the International Workshop Spatio-Temporal Database Management (STDBM'99), Edinburgh, Scotland, UK, September 10-11, pp. 39–59 (1999)
19. Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: A data model and data structures for moving objects databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, Texas, May 16-18, pp. 319–330. ACM SIGMOD Record 29(2) (2000)

20. Gargano, M., Nardelli, E., Talamo, M.: Abstract data types for the logical modeling of complex data. Information Systems **16**(6), 565–583 (1991)
21. Griffiths, T., Fernandes, A.A.A., Paton, N.W., Mason, K.T., Huang, B., Worboys, M.F.: Tripod: A comprehensive model for spatial and aspatial historical objects. In: Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001), Yokohama, Japan, November 27-30, pp. 84–102 (2001)
22. Grumbach, S., Rigaux, P., Segoufin, L.: The dedale system for complex spatial queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, June 2-4, pp. 213–224. ACM (1998)
23. Grumbach, S., Rigaux, P., Segoufin, L.: Manipulating interpolated data is easier than you thought. In: Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt, September 10-14, pp. 156–165 (2000)
24. Güting, R.H.: Geo-relational algebra: A model and query language for geometric database systems. In: Proceedings of the International Conference on Extending Database Technology (EDBT'88), Venice, Italy, March 14-18, pp. 506–527 (1988)
25. Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Transactions on Database Systems **25**(1), 1–42 (2000)
26. Güting, R.H., Schneider, M.: Realm-based spatial data types: The rose algebra. VLDB Journal **4**, 100–143 (1995)
27. Hadzilacos, T., Tryfona, N.: Logical data modeling for geographical applications. International Journal of Geographical Information Science **10**(2), 179–203 (1996)
28. Intergraph Corporation: Working with Geomedia Proffesional (2002)
29. International Business Machines Corporation (IBM): IBM DB2 Spatial Extender User's Guide and Reference, Version 7 (2001)
30. International Business Machines Corporation (IBM): IBM Informix Spatial DataBlade Module User's Guide, Version 8.11 (2001)
31. International Business Machines Corporation (IBM): Informix Geodetic DataBlade Module User's Guide, Version 3 (2001)
32. International Organization for Standardization (ISO): More Elements of Type PERIOD, Expert's Contribution (1996). ISO/IEC JTC 1/SC 21/WG 3: MCI-044
33. International Organization for Standardization (ISO): Periods of Integers, Expert's Contribution (1996). ISO/IEC JTC 1/SC 21/WG 3: MAD-151
34. International Organization for Standardization (ISO): SQL3 Part 7: Temporal, Working Draft (1996). ISO/IEC JTC 1/SC 21/WG 3: MCI-009
35. International Organization for Standardization (ISO): Information technology – Database languages – SQL multimedia and application packages – Part 3: Spatial (2003). ISO/IEC 13249-2:2000/Cor 1:2003 ISO/IEC 13249-3:2003
36. Kemp, Z., Kowalczyk, A.: Incorporating the temporal dimension in a gis. In: M.F. Worboys (ed.) Innovations in GIS 1, pp. 89–102. Taylor&Francis, London, UK (1994)
37. Kuper, G.M., Ramaswamy, S., Shim, K., Su, J.: A constraint-based spatial extension to sql. In: Proceedings of the 6th international symposium on Advances in Geographic Information Systems (GIS'98), Washington, DC, November 6-7, pp. 112–117 (1998)
38. Larue, T., Pastre, D., Viémont, Y.: Strong integration of spatial domains and operators in a relational database system. In: Proceedings of the 3rd International Symposium on Large Spatial Databases (SSD'93), Singapore, June 23-25, pp. 53–72 (1993)
39. Lorentzos, N.A., Darwen, H.: Extension to sql2 binary operations for temporal data. In: Proceedings of the 3rd HERMIS Conference, Athens, Greece, September 26-28, pp. 462–469 (1996). Invited paper
40. Lorentzos, N.A., Mitsopoulos, Y.G.: Sql extension for interval data. IEEE Transactions on Knowledge and Data Engineering **9**(3), 480–499 (1997)
41. Lorentzos, N.A., Poulovassilis, A., Small, C.: Manipulation operations for an interval-extended relational model. Data and Knowledge Engineering **17**, 1–29 (1995)
42. Lorentzos, N.A., Tryfona, N., Viqueira, J.R.R.: Relational algebra for spatial data management. In: Proceedings of the International Workshop Integrated Spatial Databases, Digital Images and GIS, Portland, Maine, June 14-16, pp. 192–208 (1999)
43. Moreira, J., Ribeiro, C., Abdessalem, T.: Query operations for moving objects database systems. In: Proceedings of the 8th ACM Symposium on Advances in Geographic Information Systems (GIS 2000), Washington, DC, November 10-11, pp. 108–114 (2000)
44. Navathe, S.B., Ahmed, R.: Temporal extensions to the relational model and sql. In: A.U. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass (eds.) Temporal Databases: Theory, Design and Implementation, pp. 6–27. Benjamin/Cummings, Redwood City, CA (1993)
45. Open GeoSpatial Consortium (OGC): Simple Features Specification for SQL 1.1 (SFS) (1999). OGC Project Document 99-049
46. Oracle corporation: Oracle Spatial: Users Guide and Reference. Release 8.1.7 (2000)
47. Park, K., Lee, J., Lee, K., Ahn, K., Lee, J., Kim, J.: The development of geus: A spatial dbms tightly integrated with an object-relational database engine. In: Proceedings of the Annual Conference Urban & Regional Information Systems Association (URISA'98), Charlotte, North Carolina, July, pp. 256–267 (1998)
48. Pavlidis, T.: Algorithms for graphics and image processing. Computer Science Press (1982)
49. PostgreSQL Global Development Group: PostgreSQL 7.2 User's Guide (2001)
50. Roussopoulos, N., Faloutsos, C., Sellis, T.K.: An efficient pictorial database system for psql. IEEE Transactions on Software Engineering **14**(5), 639–650 (1988)
51. Scholl, M., Voisard, A.: Thematic map modeling. In: Proceedings of the 1st International Symposium on Large Spatial Databases (SSD'89), Santa Barbara, California, July 17-18, pp. 167–190 (1989)
52. Scholl, M., Voisard, A.: Object-oriented database systems for geographic applications: An experiment with o2. In: F. Bancilhon, C. Delobel, P.C. Kanellakis (eds.) Building an Object-Oriented Database System, The Story of O2, pp. 585–618. Morgan Kaufmann, San Fransisco (1992)
53. Sistla, P., Wolfson, O., Chamberlain, S., Dao, S.: Modeling and querying moving objects. In: Proceedings of the 13th International Conference on Data Engineering (ICDE'97), Birmingham, UK, April 7-11, pp. 422–432 (1997)
54. Svensson, P., Huang, Z.: Geo-sal – a query language for spatial data analysis. In: Proceedings of the 2nd International Symposium on Large Spatial Databases (SSD'91), Zrich, Switzerland, August 28-30, pp. 119–140 (1991)
55. Tansel, A.U.: Adding time dimension to relational model and extending relational algebra. Information Systems **11**(4), 343–355 (1990)
56. Tomlin, C.D.: Geographic Information Systems and Cartographic Modeling. Prentice Hall, Englewood Cliffs, NJ (1990)
57. Tryfona, N., Hadzilacos, T.: Logical data modelling of spatio temporal applications: Definitions and a

model. In: Proceedings of the 1998 International Database Engineering and Applications Symposium (IDEAS 1998), Cardiff, Wales, UK, July 8-10, pp. 14–23 (1998)

58. van Roessel, J.W.: An integrated point-attribute model for four types of areal gis features. In: Proceedings of the 6th International Symposium on Spatial Data Handling (SDH'94), Edinburg, Scotland, UK, pp. vol. 1 127–144 (1994)
59. Vijlbrief, T., van Oosterom, P.: The geo++ system: An extensible gis. In: Proceedings of the 5th International Symposium on Spatial Data Handling (SDH'92), Charleston, South Carolina, August 3-7, pp. 40–50 (1992)
60. Viqueira, J.R.R.: Relational algebra for spatio-temporal data management. In: Proceedings of the EDBT 2000 PhD Workshop, Konstanz, Germany, March 31 - April 1, pp. 43–46 (2000)
61. Viqueira, J.R.R.: Formal extension of the relational model for the management of spatial and spatio-temporal data. Ph.D. thesis, Computer Science Department, University of A Coruña, Spain (2003)
62. Viqueira, J.R.R., Lorentzos, N.A.: Spatio-temporal sql. In: Y. Manolopoulos, S. Evripidou, A. Kakas (eds.) Advances in Informatics – Post-proceedings 8th Panhellenic Conference in Informatics, pp. 50–63. Lecture Notes in Computer Science 2563 Springer-Verlag, Berlin (2003)
63. Viqueira, J.R.R., Lorentzos, N.A., Brisaboa, N.R.: Management of continuous spatial changes. In: Proceedings of the 9th Panhellenic Conference on Informatics, Salonica, Greece, November 21-23, pp. 431–445 (2003)
64. Voigtmann, A.: An object-oriented database kernel for spatio-temporal geo-applications. Ph.D. thesis, Westf. Wilhelms-Universitt Mnster, Germany (1997)
65. Winter, S., Frank, A.U.: Topology in raster and vector representation. GeoInformatica **4**(1), 35–65 (2000)
66. Worboys, M.F.: A unified model for spatial and temporal information. The Computer Journal **37**(1), 36–34 (1994)
67. Yeh, T., de Cambray, B.: Modeling highly variable spatio-temporal data. In: Proceedings of the 6th Australasian Database Conference (ADC'95), Glenelg/Adelaide, South Australia, January, pp. Vol 17, No. 2, 221–230 (1995)

Appendix

General Remarks

It is assumed that the SQL:1999 documentation is available for reference, hence only its extension is given here, in **bold**. Note also that additional constructs, which relate only to the management of time and intervals, can be found in [40].

Query Expressions

<query expression> ::=
  <IXSQL non–join query expression>
  | **<IXSQL joined table>**
  | **<IXSQL unary query expression>**

<IXSQL non–join query expression> ::=
  (<non-join query expression>) [**<reformat clause>**]
    [**<normalise clause>**]
  | <non–join query expression>

<non–join query expression> ::=
  <non–join query term>
  | <query expression> UNION [ALL]
    [**EXPANDING** (**<reformat column list>**)]
    [<corresponding spec>]
    <IXSQL query term>
  | <query expression> EXCEPT [ALL]
    [**EXPANDING** (**<reformat column list>**)]
    [<corresponding spec>]
    <IXSQL query term>
  | <query expression> **WUNION** [**ALL**]
    OF (**<reformat column list>**)
    [**EXPANDING** (**<reformat column list>**)]
    <IXSQL query term>
  | <query expression> **WEXCEPT** [**ALL**]
    OF (**<reformat column list>**)
    [**EXPANDING** (**<reformat column list>**)]
    <IXSQL query term>

<IXSQL query term> ::=
  <IXSQL non–join query term>
  | **<IXSQL joined table>**
  | **<IXSQL unary query expression>**

<IXSQL non–join query term> ::=
  <non–join query term>
  | (<non–join query term>) [**<reformat clause>**]
    [**<normalise clause>**]

<non–join query term> ::=
  <non–join query primary>
  | **<IXSQL query term>** INTERSECT [ALL]
    [**EXPANDING**(**<reformat column list>**)]
    [<corresponding spec>]
    <IXSQL query primary>
  | **<IXSQL query term> WINTERSECT** [**ALL**]
    OF (**<reformat column list>**)
    [**EXPANDING**(**<reformat column list>**)]
    <IXSQL query primary>

<IXSQL query primary> ::=
  <non–join query primary>
  | **<IXSQL joined table>**
  | **<IXSQL unary query exp>**

<non–join query primary> ::=
  <simple table>
  | (**<IXSQL non–join query expression>**)

<simple table> ::=
  <IXSQL query specification>
  | **<IXSQL table value constructor>**
  | **<IXSQL explicit table>**

<IXSQL query specification> ::=
  <query specification>[**<reformat clause>**]
    [**<normalise clause>**]

<query specification> ::=
  SELECT <set quantifier> <select list>
    <table expression>

<table expression> ::=
  <from clause> [<where clause>]
    [<group by clause>] [<having clause>]

<from clause> ::=
  FROM <table reference> [{, <table reference>}. . .]

<table reference> ::=
  <table name> [[AS] <correlation name>
    [(derived column list)]]
  | <table subquery> [[AS]
    <correlation name> [(derived column list)]]
  | **<IXSQL joined table>**
  | **<IXSQL unary query expression>**

<IXSQL row value constructor> ::=

(<row value constructor>)
[**<reformat clause>**][**<normalise clause>**]
<IXSQL table value constructor> ::=
(<table value constructor>)
[**<reformat clause>**] [**<normalise clause>**]
<IXSQL explicit table>::=
TABLE <table name>
[**<reformat clause>**] [**<normalise clause>**]
<IXSQL joined table> ::=
(<joined table>) [**<reformat clause>**]
[**<normalise clause>**]
| <joined table>
<joined table> ::=
<cross join>
| <qualified join>
| **<IXSQL overlay>**
| (<joined table>)
<cross join> ::=
<table reference> CROSS JOIN
[**EXPANDING** (**<reformat column list>**)]
<table reference>
<qualified join> ::=
<table reference> [NATURAL] [<join type>] JOIN
[**EXPANDING** (**<reformat column list>**)]
<table reference>
[<join specification>]
<IXSQL overlay> ::=
<table reference> [**NATURAL**][**<overlay type>**]
OVERLAY [**ALL**]
[**OF** (**<reformat column list>**)]
[**EXPANDING**(**<reformat column list>**)]
<table reference>
<overlay type> ::=
INNER
| {**LEFT** | **RIGHT** | **FULL**} [**OUTER**]
<IXSQL unary query expression> ::=
<unary query expression>
| (**<unary query expression>**)
[**<reformat clause>**]
[**<normalise clause>**]
<unary query expression> ::=
(**<unary query expression>**)
| <table reference> {**COMPLEMENTATION**
| **BOUNDARY**
| **ENVELOPE** [**ALL**]}
OF (**<reformat column list>**)
[**EXPANDING** (**<reformat column list>**)]
| <table reference> **BUFFER** [**ALL**]
OF (**<reformat column list>**)
WITHIN DISTANCE
(**<reformat column list>**)
[**EXPANDING** (**<reformat column list>**)]
<reformat clause> ::=
REFORMAT AS <reformat item>
<reformat item> ::=
FOLD [**ALL**] **<reformat column list>**
[**<reformat item>**]
UNFOLD [**ALL**] **<reformat column list>**
[**<reformat item>**]
<normalise clause> ::=
NORMALISE ON [**ALL**]
<reformat column list>
<reformat column list> ::=
<reformat column> [{, **<reformat column>**}. . .]
<reformat column> ::=
<column reference> | <unsigned integer>

Notes: The keyword **EXPANDING** has been borrowed from [39].

Literals
<IXSQL literal> ::=
<literal>
| **<IXSQL period literal>**
<IXSQL period literal> ::=
[<literal>, <literal>]
<literal> ::=
<signed numeric literal>
| <general literal>
<general literal> ::=
<character string literal>
| <national character string literal>
| <bit string literal>
| <hex string literal>
| <datetime literal>
| <interval literal>
| **<IXSQL spatial literal>**
<IXSQL spatial literal> ::=
<spatial object type> <spatial quanta string>
<spatial object type>::=
POINT | **PLINE** | **LINE** |
SURFACE | **SURFACE**
<spatial quanta string> ::=
'**<spatial quantum>**[{**<spatial quantum>**}. . .]'
<spatial quantum> ::=
<spatial quantum type> <unsigned integer>
<spatial quantum type> ::= **P** | **H** | **V** | **S**

Search Conditions
<IXSQL predicate> ::=
<predicate>
| **<IXSQL period predicate>**
| **<IXSQL spatial predicate>**
<IXSQL period predicate> ::=
<IXSQL period binary predicate>
<IXSQL period binary predicate> ::=
<IXSQL period value expression>
<period binary predicate name>
<IXSQL period value expression>
<IXSQL spatial predicate> ::=
<IXSQL spatial binary predicate>
| **<IXSQL spatial n–ary predicate>**
<IXSQL spatial binary predicate> ::=
<IXSQL spatial value expression>
<spatial binary predicate name>
<IXSQL spatial value expression>
<spatial binary predicate name> ::=
cp | **disjoint** | **surrounds**
<IXSQL spatial n–ary predicate> ::=
is_point (**<IXSQL patial value expression>**)
| **is_pure_line** (**<IXSQL spatial value expression>**)
| **is_line** (**<IXSQL spatial value expression>**)
| **is_pure_surface**
(**<IXSQL spatial value expression>**)
| **is_surface**(**<IXSQL spatial value expression>**)
| **is_hybrid_surface**
(**<IXSQL spatial value expression>**)
| **is_simple**(**<IXSQL spatial value expression>**)
| **is_circular**(**<IXSQL spatial value expression>**)
| **conductive**
(**<IXSQL spatial value expression>**,
<IXSQL spatial value expression>,
<IXSQL spatial value expression>,
<IXSQL spatial value expression>)
| **has_holes**(**<IXSQL spatial value expression>**)

Notes: Both <comparison predicate>s (=, <>, <, >, etc.) and <quantified comparison predicate>s (= SOME, > ANY, etc.) of SQL:1999 can be applied to pairs of <row value constructor>s containing some <value expression> of either a period or a spatial type.

Value Expressions

```
<value expression> ::=
    <atomic value expression>
    | <IXSQL period value expression>

<atomic value expression> ::=
    <numeric value expression>
    | <string value expression>
    | <datetime value expression>
    | <interval value expression>
    | <IXSQL spatial value expression>

<IXSQL common primary> ::=
    <common primary>
    | <IXSQL period common primary>

<IXSQL numeric primary> ::=
    <numeric primary>
    | <IXSQL period numeric primary>
    | | <IXSQL spatial numeric primary>

<IXSQL spatial numeric primary> ::=
    ord(<IXSQL spatial value expression>)
    | h_coord(<IXSQL spatial value expression>)
    | v_coord(<IXSQL spatial value expression>)
    | distance
        (<IXSQL spatial value expression>,
        <IXSQL spatial value expression>)
    | greatest_distance
        (<IXSQL spatial value expression>,
        <IXSQL spatial value expression>)
    | length(<IXSQL spatial value expression>)
    | area(<IXSQL spatial value expression>)

<IXSQL datetime primary> ::=
    <datetime primary>
    | <IXSQL period datetime primary>

<IXSQL spatial value expression> ::=
    form_point
        (<numeric value expression>,
        <numeric value expression>)
    | to_point(<IXSQL spatial value expression>)
    | to_pure_line(<IXSQL spatial value expression>)
    | to_line(<IXSQL spatial value expression>)
    | to_pure_surface
        (<IXSQL spatial value expression>)
    | to_surface(<IXSQL spatial value expression>)
```