How Does the Spatial Data Redundancy Affect Query Performance in Geographic Data Warehouses?

Rodrigo Costa Mateus[1], Thiago Luís Lopes Siqueira[2],
Valéria Cesário Times[1], Ricardo Rodrigues Ciferri[3], Cristina Dutra de Aguiar Ciferri[4]

[1] Informatics Center, Federal University of Pernambuco, 50733-970, Recife, PE, Brazil
{rcm3,vct}@cin.ufpe.br
[2] São Paulo Federal Institute of Education, Science and Technology, 13565-905, São Carlos, SP, Brazil
prof.thiago@cefetsp.br
[3] Department of Computer Science, Federal University of São Carlos, 13565-905, São Carlos, SP, Brazil
ricardo@dc.ufscar.br
[4] Department of Computer Science, University of São Paulo, 13560-970, São Carlos, SP, Brazil
cdac@icmc.usp.br

Abstract. Geographic Data Warehouses (GDWs) are traditional data warehouses with spatial attributes that are used for defining spatial dimension tables, spatial measures and spatial hierarchies. Non-redundant spatial data warehouse schemas have been recognized as an essential issue in the GDW design. Although the lack of spatial redundancy represents a gain in data storage, it implies in a need for performing expensive join operations to answer a given query that may refer to one or more query windows. In this paper, we investigate to what extent the separate storage of spatial and conventional data is recommended in GDW, according to increasing numbers of query windows. We also investigate if the complexity of the spatial data (i.e. points versus polygons) influences the choice of storing spatial and conventional data in the same or in different dimension tables. Our experimental results indicated that if non-redundant spatial data are represented as point objects, an approach to avoid additional join costs by storing both point data and their descriptive data in a single table should be chosen. The results also showed that redundant GDW schemas introduce a severe drawback, as some spatial analytical queries cannot reuse previously fetched spatial data, impairing query performance. Finally, based on the experimental results, we propose in this paper a set of guidelines for the design of logical GDW schemas, called "Logical GDW Design Guidelines".

Categories and Subject Descriptors: H.2.1 [**Information Systems**]: Logical Design—*data models*

Keywords: benchmark, geographic data warehouse, performance evaluation

1. INTRODUCTION

There are several approaches for handling business applications that help users in increasing the productivity of decision-making processes, such as Data Warehouse (DW), On-Line Analytical Processing (OLAP) and Geographic Information System (GIS). A DW is a multidimensional database that stores subject-oriented, integrated, time-variant and non-volatile data, and is often modeled through a star schema composed of fact and dimension tables [Kimball and Ross 2002]. An OLAP tool is a software aimed at multidimensional processing of data extracted from DWs, allowing these data to be analyzed in different perspectives and levels of aggregation [Chaudhuri and Dayal 1997]. Regarding GIS, it focuses on the acquisition, manipulation, visualization and analysis of spatial objects [Câmara et al. 1996; Demers 2000]. The integration of the features of OLAP tools and GIS has been referred to as spatial OLAP (SOLAP) [da Silva et al. 2010], and benefits the strategic decision-making process by broadening the use of spatial predicates in OLAP queries over a Geographic DW (GDW). A GDW,

therefore, extends a DW by storing vector geometries in spatial attributes and by defining spatial dimension tables, spatial measures and spatial hierarchies [Siqueira et al. 2009b].

Concerning schemas of GDWs, GeoDWFrame is a framework based on the star schema that consists of a set of concepts and principles for guiding the design of GDWs [Fidalgo et al. 2004]. Based on this framework and using OCL (Object Constraint Language) restrictions, a UML (Unified Modeling Language) class diagram of a GDW metamodel was introduced in [Fonseca et al. 2007]. This GDW metamodel, which has formally been defined in [Times et al. 2008], aids the specification of GDW schemas by *avoiding spatial data redundancy* and storing spatial and conventional attributes separately, even though this storage introduces additional joins among conventional and spatial dimension tables to process SOLAP queries. Also, redundant and non-redundant GDW schemas were compared and experimental results indicated that *spatial redundancy is related to high performance losses* both in query processing and in storage requirements [Siqueira et al. 2009b]. Therefore, the literature clearly recommends *the use of non-redundant schemas* in GDWs.

There are three main operations related to SOLAP query processing performance in a star schema: (i) joining large fact tables and spatial and conventional dimension tables; (ii) computing one or more costly spatial predicates based on spatial ad hoc query windows; and (iii) aggregating data according to different spatial granularity levels. The aforementioned previous works recommend the use of non-redundant schemas based on the claim that computing *few* additional joins is less costly than storing a large amount of redundant spatial data in the dimension table and processing them to answer SOLAP queries. However, we argue that, if the number of query windows found in SOLAP queries is increased, spatial redundancy avoidance requires performing *several* joins to answer these queries. In this paper, we go one step forward to previous works by investigating to what extent the separate storage of spatial and conventional attributes is recommended in GDW, according to increasing numbers of query windows. This is the main contribution of this paper.

Another contribution of this paper is related to non-redundant GDW schemas. It has been recognized that spatial and conventional attributes *should not be stored in the same dimension table*, regardless of the characteristics of the spatial objects. However, in several cases, the spatial objects are characterized by being *distinct* and having a *1:1 association* with the dimension table primary key values. For instance, a non-redundant GDW that represents historical data related to orders and sales may store each customer in terms of its ID and its unique georeferenced address. In these cases, it is important to assess if the *joint storage* of spatial and conventional data that follow the mentioned characteristics really impairs SOLAP query performance. In this paper, we examine this issue and also investigate if the complexity of the spatial objects (i.e. points versus polygons) influences the choice of storing spatial and conventional attributes in the same or in different dimension tables.

The third contribution of this paper is related to the design of logical GDW schemas. Although some principles for guiding the design of GDWs have been proposed by GeoDWFrame, they do not focus on the joint storage of spatial and conventional data in a single dimension table nor on the issue of uniqueness and complexity of spatial attributes. On the other hand, in this paper we experimentally investigate these two issues. We provide a set of guidelines for the design of logical GDW schemas, called "Logical GDW Design Guidelines", that take into account the joint storage of spatial and conventional data. Another difference refers to the fact that the principles introduced in GeoDWFrame were not validated experimentally, while the guidelines proposed in this paper are based on experimental performance results.

The remainder of this paper is organized as follows. Section 2 describes the basic concepts used throughout the paper. The contributions of the paper are described in Sections 3 to 5, as follows. Section 3 compares non-redundant and redundant GDW schemas concerning increasing numbers of spatial query windows, Section 4 analyses the joint storage of spatial and conventional attributes in non-redundant GDW schemas, and Section 5 describes the proposed "Logical GDW Design Guidelines". Section 6 surveys related work, while Section 7 concludes the paper and highlights future

work.

2. THEORETICAL FOUNDATION

A conventional DW is often implemented in relational databases through a star schema, which is composed of fact and dimension tables [Kimball and Ross 2002]. Fact tables store numeric measures of interest, while dimension tables contain attributes that contextualize these measures. Each attribute of a dimension may have a relationship with other attributes of the same dimension through hierarchies of attributes, which specify levels of aggregation and, consequently, data granularity. For instance, the dimension table *Customer* may be described by the attributes *ID*, *address*, *city*, *nation* and *region*, so that $region \preceq nation \preceq city \preceq address$. In this hierarchy, *region* is the attribute of the highest level of granularity, while *address* is the attribute of the lowest level of granularity. In addition, according to Harinarayan (1996), $v_1 \preceq v_2$ if and only if it is possible to answer v_1 using no more than the results of v_2. Therefore, it is possible to obtain the measures for higher granularity attributes by aggregating the results from lower granularity attributes.

Based on hierarchies of attributes, data in a DW are usually organized from a lower level of granularity that contains detailed data up to a higher level of granularity containing highly summarized data. There also may be several intermediate levels, which represent increasing levels of aggregation. Furthermore, this organization in levels of aggregation allows decision-making users to begin their analyses at a higher level of granularity to look over broad perspectives and then progressively access less aggregated levels as more specific data are required. This kind of analysis illustrates drill-down queries. On the other hand, roll-up queries investigate data in progressively less detailed aggregation levels. For instance, the following queries could be commonly required in a drill-down analysis: *revenue by region*, then *revenue by nation*, then *revenue by city*, and then *revenue by address*, where *revenue* is a numeric measure.

Differently from a conventional DW, a GDW stores spatial data as specific attributes in dimension tables or as spatial measures in fact tables [Malinowski and Zimányi 2004; 2006; 2008]. In a GDW, spatial hierarchies may be defined over spatial attributes of one or more dimension tables. A spatial hierarchy is typically a *1:N* association among higher and lower granularity spatial attributes that is determined by a spatial relationship, such as $region\_geo \preceq nation\_geo \preceq city\_geo \preceq s\_address\_geo$, where the suffix *_geo* represents spatial attributes that store vector geometries and the spatial relationship is containment.

A GDW can be organized into a star schema according to two different approaches: redundant GDW schema and non-redundant GDW schema. In a redundant GDW schema, the dimension tables store both conventional and spatial attributes. Therefore, a given dimension table may replicate the same spatial data several times (e.g. the geometry of a given nation). Figure 1 depicts an example of a redundant GDW schema, adapted from the Star Schema Benchmark (SSB) [O'Neil et al. 2009] and proposed in [Siqueira et al. 2009b]. In this paper, we call this schema Geographic Redundant Star Schema Benchmark (GRSSB). The spatial attributes in the GRSSB schema are *s_address_geo*, *s_city_geo*, *s_nation_geo* and *s_region_geo* for the dimension table *Supplier*, and *c_address_geo*, *c_city_geo*, *c_nation_geo* and *c_region_geo* for the dimension table *Customer*. The changes that were performed preserved the original SSB conventional data and created a spatial hierarchy based on the previously defined conventional dimensions. Both the dimension tables *Supplier* and *Customer* have the following hierarchy: $region\_geo \preceq nation\_geo \preceq city\_geo \preceq address\_geo$.

On the other hand, the non-redundant GDW schema uses two types of dimension tables. The spatial dimension table stores the ID and the vector geometry of each spatial data, while the corresponding conventional dimension table contains the conventional attributes and a foreign key to the spatial dimension table. For instance, Figure 2 shows an example of a non-redundant GDW schema, where spatial data related to *Customer* is stored in the spatial dimension table *C_Address*

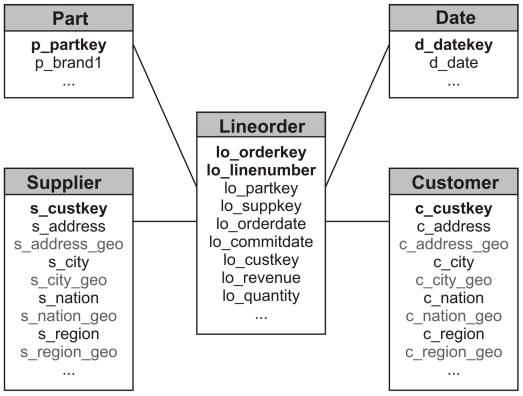

Fig. 1. The redundant GDW schema (called the GRSSB schema).

(i.e. *c_ address_ pk* is the ID and *c_ address_ geo* is the vector geometry). The conventional dimension table *Customer* has the attribute *c_ address_ fk* as the foreign key to the table *C_ Address*. In this non-redundant GDW schema, the dimension tables *Customer* and *Supplier* share *City*, *Nation* and *Region* locations, but not *Address* locations. Therefore, only the domains of the attributes *s_ address_ geo* and *c_ address_ geo* are disjoint, while the domains of the attributes *city_ geo*, *nation_ geo* and *region_ geo* are overlapping for customers and suppliers. In this paper, we call the schema of Figure 2 Geographic Hybrid Star Schema Benchmark (GHSSB), which was proposed in [Siqueira et al. 2009b]. We used the term *Hybrid* to comply with the GeoDWFrame dimension tables definitions [Fidalgo et al. 2004] as both the dimension tables *Supplier* and *Customer* found in this schema are of type hybrid, which deals with location descriptions and conventional data (e.g. customer's address plus name and phone number).

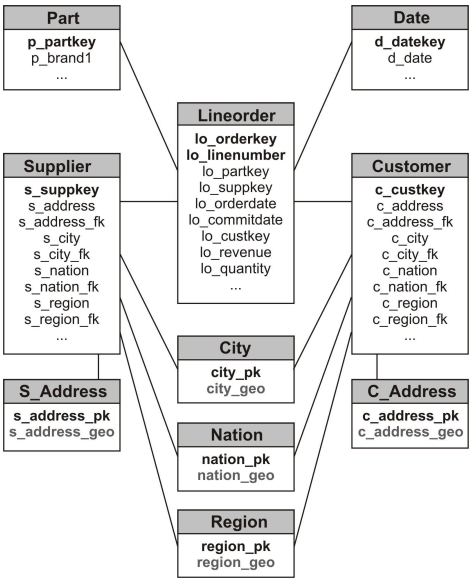

Fig. 2. The non-redundant GDW schema (called the GHSSB schema).

The type of geographic query investigated in this paper is known as range query [Gaede and Günther 1998]. Given a bi-dimensional rectangle *QW* (i.e. query window) whose sides are parallel to the axes of their respective dimensions, this query finds all the objects that satisfy a topological relationship with

respect to QW. Each type of topological relationship (e.g. intersects, contains, within) characterizes a specific subtype of a range query. Conventional and spatial hierarchies enable the processing of drill-down and roll-up operations extended with range queries [Rivest et al. 2005; Malinowski and Zimányi 2008]. In these operations, the spatial ad hoc query window is a region that was not previously stored in a spatial dimension table. For instance, the following queries could be commonly required in a drill-down analysis extended with range queries: *revenue by c_region_geo that intersects a given spatial ad hoc query window* QW_1, then *revenue by c_nation_geo that intersects a given spatial ad hoc query window* QW_2, then *revenue by c_city_geo that intersects a given spatial ad hoc query window* QW_3, and then *revenue by c_address_geo that is within a given spatial ad hoc query window* QW_4. These queries are further explained in Section 3.1.

3. PERFORMANCE TESTS USING REDUNDANT AND NON-REDUNDANT GDW SCHEMAS

In this section, we show and discuss performance results related to the experiments based on both the redundant GDW schema (i.e. the GRSSB schema) and the non-redundant GDW schema (i.e. the GHSSB schema). The goal of these performance tests is to investigate to what extent the separate storage of conventional and spatial attributes in dimension and spatial dimension tables is recommended in GDWs, according to increasing numbers of query windows and, consequently, increasing numbers of joins among these tables.

This section is organized as follows. Section 3.1 describes the experimental setup that was used to investigate how much SOLAP query performance was affected by increasing numbers of query windows, Section 3.2 details the performance results for *disjoint* query windows, and Section 3.3 describes the obtained measurements for *overlapping* query windows.

3.1 Experimental Setup

The data generation for the GHSSB schema was performed as follows. The GHSSB schema was created using the SSB benchmark with scale factor 10, occupied 15 GB and generated 60 million of tuples in the fact table, 5 distinct regions, 5 nations per region, 10 cities per nation and a certain quantity of addresses per city that varied from 349 to 455. Cities, nations and regions were represented by polygons, while addresses were represented by points. Polygons were real-world data adapted from Tiger/Line (*www.census.gov/geo/www/tiger*), while points were synthetic data.

Regarding the data generation for the GRSSB schema, it was carried out similarly to the data generation of the GHSSB schema. It produced the same number of tuples for fact and dimension tables, data was generated using the same scale factor of the SSB benchmark and spatial data had the same geometric types and values. But in the GRSSB schema, the vector geometries were stored redundantly and, as a result, the GRSSB schema stored about 150 GB. For instance, the polygon representing the map of Brazil was stored in every row whose customer or supplier was located in Brazil.

With regard to the workload, we used roll-up queries extended with range queries. These SOLAP queries were based on predefined spatial attribute hierarchies of dimensions tables and spatial dimension tables that encompassed several granularity levels (i.e. address, city, nation and region). We replaced the conventional predicate of SSB queries with spatial predicates involving query windows. The number of query windows were one, two, four and eight and, for each number of query windows, the experiments were performed by submitting five complete SOLAP roll-up queries to both the redundant and the non-redundant GDW schemas and taking the average of the measurements for each granularity level.

While the SOLAP roll-up queries that referred to a single query window were based on Query Q2.3 from the SSB benchmark (Figure 3), the queries with two, four and eight query windows were

based on Query Q3.3 from this benchmark (Figure 4). For queries with one and two query windows, the windows were defined as follows. They were quadratic, had a correlated distribution with the spatial data and their sizes were proportional to the spatial granularity. Addresses were evaluated with containment range queries and their query windows covered 0.001% of the extent, while cities, nations and regions were evaluated with intersection range queries and their query windows covered 0.05%, 0.1% and 1% of the extent, respectively. The centroids of the query windows consisted of random addresses, where one of them was a customer address and, for two query windows, the other query window was a supplier address also randomly chosen.

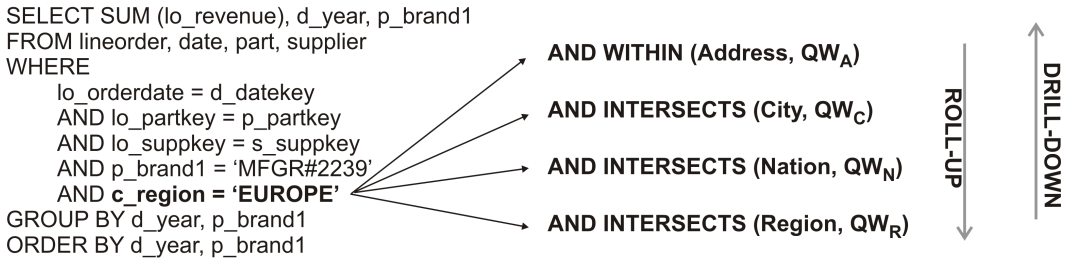

Fig. 3. Query Q2.3 from the SSB benchmark adapted to a single query window QW.

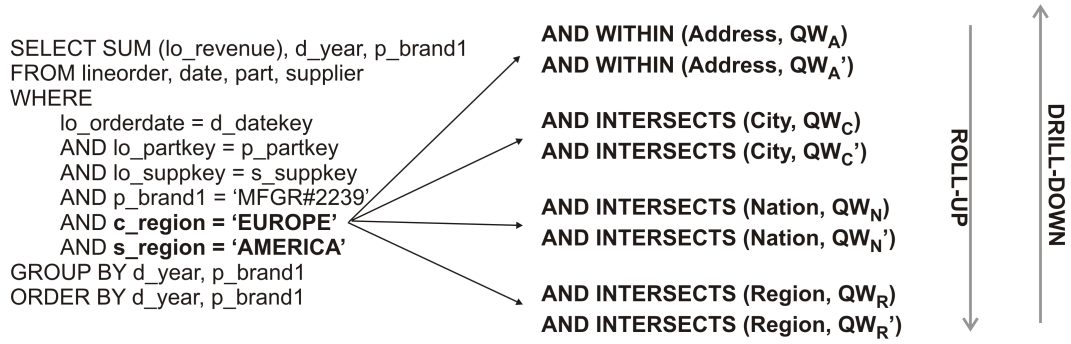

Fig. 4. Query Q3.3 from the SSB benchmark adapted to two query windows QW and QW'.

Due to some restrictions found in the SSB schema, which were related to its total number of spatial dimensions, it was necessary to adapt the Query Q3.3 from SSB to investigate the effect of performing increasing numbers of query windows (i.e. specifically aiming at using four and eight query windows). For this purpose, we used the set operator UNION of SQL. This operator was used to combine the partial results derived from the application of a pair of query windows with the two spatial dimensions of the adapted SSB schema (i.e. the dimension tables *Customer* and *Supplier*).

Experiments were conducted on a computer with 2.8 GHz Pentium D processor, 8 GB of main memory, 7200 RPM SATA 320 GB hard disk, Linux CentOS 5.2, PostgreSQL 8.2.5 and PostGIS 1.3.3. The star-join computations were aided by the GiST index (`http://gist.cs.berkeley.edu`) defined over the spatial attributes. In the performance tests, we gathered the elapsed time in seconds.

3.2 Performance Evaluation for Disjoint Query Windows

In this section, we discuss the performance results for all the granularity levels considering an increasing number of *disjoint* query windows. Table I shows the performance results for SOLAP roll-up queries related to a single query window, while Tables II, III and IV show the response times of SOLAP roll-up queries having two, four and eight query windows, respectively.

Our main performance findings are shown in the reduction columns, which compare how much faster the SOLAP roll-up queries were processed over the GHSSB schema than over the GRSSB schema. In general, the greater the number of query windows in these queries, the greater the processing costs were, once this implied in a need for processing additional joins. There were only the following exceptions regarding two query windows. For the city and nation granularity levels, the elapsed times of the queries were almost the same for a single query window and for two query windows. This fact is highlighted in gray in Tables I and II. We can conclude that the increase in the number of query windows in these cases did not impair the query performance.

Comparing the GRSSB schema with the GHSSB schema, the latter significantly improved the query processing performance, despite the fact that it required more joins because the spatial data were stored separately from their corresponding dimention tables. The GHSSB schema produced a performance gain that ranged from at least 19.73% up to 99.97%. The smallest performance gain (i.e. 19.73%) was obtained at the address granularity level. This is due the fact that this level stored point data which were both less computationally expensive to evaluate spatial relationships and not redundant in the dimension table.

Granularity Level	**Query Processing (elapsed time in seconds)**		**Reduction (%)**
	GHSSB	**GRSSB**	
ADDRESS	146.22	182.19	19.74
CITY	94.52	195.43	51.64
NATION	93.66	999.69	90.63
REGION	93.42	3,523.27	97.35

Table I. Performance results for a single disjoint query window.

Granularity Level	**Query Processing (elapsed time in seconds)**		**Reduction (%)**
	GHSSB	**GRSSB**	
ADDRESS	180.01	6,267.19	97.13
CITY	93.05	111,600.00	99.92
NATION	93.26	86,400.00	99.89
REGION	110.43	86,400.00	99.87

Table II. Performance results for two disjoint query windows.

Granularity Level	**Query Processing (elapsed time in seconds)**		**Reduction (%)**
	GHSSB	**GRSSB**	
ADDRESS	266.16	12,140.65	97,81
CITY	160.99	25,200.00	99.36
NATION	141.03	23,400.00	99.40
REGION	194.69	39,600.00	99,51

Table III. Performance results for four disjoint query windows.

3.3 Performance Evaluation for Overlapping Query Windows

The use of *overlapping* query windows aims at evaluating the reuse of previously fetched spatial objects when performing the spatial predicate computation in SOLAP roll-up queries. Therefore, we established in our tests an overlapping threshold of 50%. This means that, if ten objects satisfy the spatial predicate for a given query window, namely QW_i, then five of them must also satisfy it for the subsequent query window QW_{i+1}. It is not required that all of these five objects be referenced in the fact table, but it is expected that at least two of them are in the fact table to satisfy the query related to these query windows. The overlapping properties are valid for all query windows, for all the granularity levels and for queries related to *Customer* and *Supplier*.

Granularity Level	Query Processing (elapsed time in seconds)		Reduction (%)
	GHSSB	GRSSB	
ADDRESS	479.38	31,884.02	98.50
CITY	302.72	37,800.00	99.20
NATION	328.28	41,700.00	99.21
REGION	442.98	43,200.00	98.98

Table IV. Test results for eight disjoint query windows.

Tables V, VI and VII show the response times of queries having two, four and eight overlapping query windows. Our experimental results indicated that at any granularity level and for any number of overlapping query windows, the spatial data redundancy drastically impaired the performance of SOLAP roll-up queries: the non-redundant GHSSB schema outperformed the redundant GRSSB schema by at least 95.52%. These results corroborate the fact that data organization affects query processing performance, which is a well-known fact in the database literature. Specially in GDWs, the storage of both multidimensional and spatial data requires the manipulation of higher volumes of data and, therefore, GDWs are even more sensitive to data organization. In addition, the organization of spatial objects in multidimensional data structures affects SOLAP query processing performance because this organization may prevent the reuse of previously fetched spatial objects and may affect the number of scanned tables in query processing.

The GRSSB schema presents a severe drawback since some queries are not able to reuse previously fetched spatial objects as illustrated as follows. Suppose that $QW_{C,City}$ and $QW_{S,City}$ are query windows for *Customer* and *Supplier*, respectively, at the city granularity level. Although $QW_{C,City}$ and $QW_{S,City}$ are overlapping query windows, they require scanning two distinct tables of the GRSSB schema, Customer and Supplier, specifically on the attributes *c_ city_ geo* and *s_ city_ geo*. Even if these two attributes have spatial indices built over them, the indices have to be scanned to filter the spatial objects before the corresponding spatial objects stored in tables be analyzed for refinement purposes. On the other hand, the GHSSB schema maintains for each level a single spatial object table with a single spatial attribute, *city_ geo*, and a single index defined on it. Therefore, queries performed against the GHSSB schema and evolving $QW_{C,City}$ and $QW_{S,City}$ require accessing a single index and a single spatial object table. This benefited the non-redundant schema over the redundant one for all the granularity levels.

Granularity Level	Query Processing (elapsed time in seconds)		Reduction(%)
	GHSSB	GRSSB	
ADDRESS	92.86	6,873.80	98.65
CITY	91.60	14,400.00	99.36
NATION	91.80	10,800.00	99.15
REGION	134.18	12,600.00	98.94

Table V. Performance results for two overlapping query windows.

Granularity Level	Query Processing (elapsed time in seconds)		Reduction(%)
	GHSSB	GRSSB	
ADDRESS	263.05	12,426.39	97.88
CITY	178.50	7,200.00	97.52
NATION	169.50	3,780.00	95.52
REGION	230.47	5,400.00	95.73

Table VI. Performance results for four overlapping query windows.

Granularity Level	Query Processing (elapsed time in seconds)		Reduction(%)
	GHSSB	GRSSB	
ADDRESS	500.03	26,400.00	98,11
CITY	344.45	7,800.00	95,58
NATION	280.99	7,200.00	96,10
REGION	426.95	9,000.00	95,26

Table VII. Performance results for eight overlapping query windows.

4. PERFORMANCE RESULTS USING ONLY NON-REDUNDANT SCHEMAS

In this section, we show and discuss performance results related to the experiments based on non-redundant GDW schemas. The goal of these tests is to assess the benefits of performing joins in a non-redundant schema by maintaining the spatial data (i.e. point data or polygons representing addresses) in separated dimension tables from their corresponding descriptive data. Descriptive data are conventional data that describe the related spatial data. For instance, an address represented as a string is a descriptive data of the related location of an address represented as a point data. Section 4.1 describes the experimental setup that was used to evaluate the performance of SOLAP queries, while Section 4.2 discusses the performance results.

4.1 Experimental Setup

Experiments were conducted using the same computer configuration and the same DBMS resources (including spatial indices) as those described in Section 3.1. Regarding the workload, it was based on Query Q2.3 from the SSB benchmark. SOLAP queries had the following properties: (i) they referred to a single query window; (ii) they were related to the lowest spatial granularity level (i.e. the address granularity level); and (iii) they retrieved non-redundant coordinate data. We replaced the conventional predicate of Query Q2.3 with the spatial predicate *within*. Each query window was quadratic and had a correlated distribution with spatial data, and its centroid was a random customer address. Its size was proportional to the lowest spatial granularity level. Therefore, addresses were evaluated with containment range queries and their query windows covered 0.001% of the extent.

We defined two test configurations based on non-redundant GDW schemas. The first one used the GHSSB schema described in Figure 2, while the second one was based on the NewGHSSB schema proposed in Figure 5. The NewGHSSB schema was designed as an adaptation of the GHSSB schema, in which for each spatial object type found in the GHSSB schema (e.g. *Customer* and *Supplier*), we added all spatial data to the corresponding dimension table if these spatial data are not likely to be shared among any other spatial dimension tables, such as the attribute *c_ address_ geo* in Figure 5. This schema adaptation aimed at avoiding unnecessary joins when space storage is not affected by spatial redundancy.

Polygon data were generated as follows. They were created from existing customer's addresses of the original GHSSB schema. For each address, the following computations were executed: (i) a squared shape was created using the customer's address as its centroid (X, Y) and an offset value denoted by d for defining its corners (i.e. $X_{MIN} = X - d$; $X_{MAX} = X + d$; $Y_{MIN} = Y - d$; $Y_{MAX} = Y + d$; where $d = 0.003$); (ii) each side of the square was divided by 100 to increase the spatial object geometry complexity by generating polygon data consisting of 400 points; and (iii) for each vertical side of the polygon, points were created from the beginning to the end of the side by shifting 0.0001 and generating 100 points on each side. In the horizontal side, the points were shifted using the same offset of 0.0001, generating 100 points. Figure 6a depicts a sample of the original addresses represented as points, while Figure 6b shows the corresponding polygons generated. Figure 6c details the geometry of a polygon, which is slightly different from a square as its sides are generated using a small shift.

The *BuildSquaredShapes* algorithm (Algorithm 1), which was developed according to the afore-

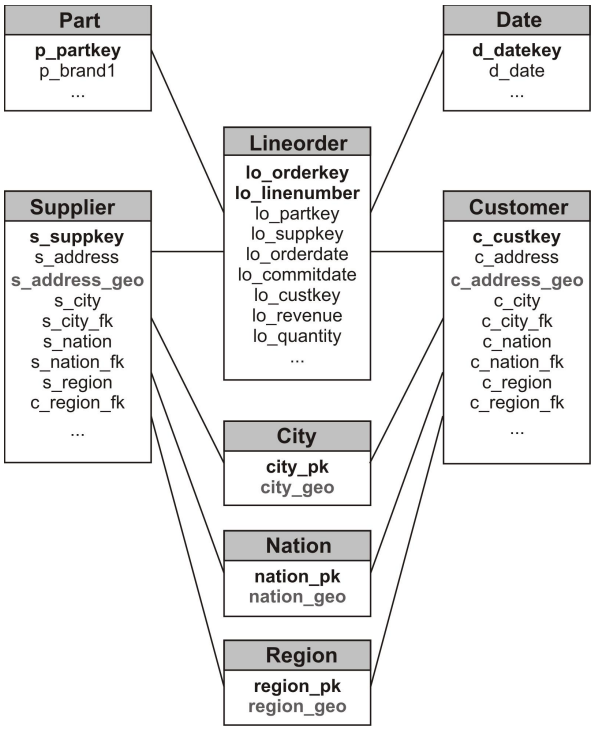

Fig. 5. The new non-redundant GDW schema (called the NewGHSSB schema).

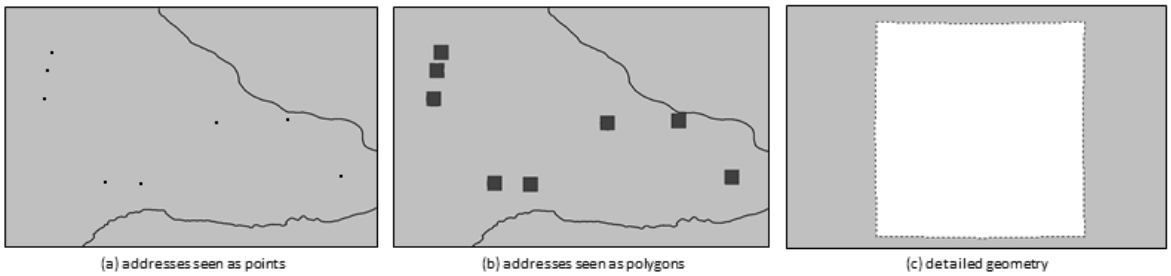

Fig. 6. The two kinds of geometry considered for customers addresses and the detailed geometry of a polygon.

mentioned discussion, works as follows. It has a single parameter $setOfPoints$ that specifies the number of points and the coordinates of each point. In lines 1 to 3, the variable *count* indicating the current number of polygons generated is initialized to zero, an offset value of 0.003 distance units is assigned to d, and the variable $setOfPolygons$ that represents the squared shapes to be returned is initialized to empty. After comparing the value of *count* with the number of elements found in the input data collection (line 4), there is a loop that initially computes the numerical values of the coordinates that describe the boundaries of a square (lines 5 to 8). These values, called $X_{MIN}, X_{MAX}, Y_{MIN}$ and Y_{MAX}, are obtained using d. Then, in line 9, a shift value is computed dividing the length of each square line segment by 100 to generate a polygon with 400 points. In line 10, a new polygon object is created and, in lines 11 to 14, each time the algorithm $generatePoints$ is called, 100 points are generated for each side of the polygon and are added to the set of points of this new polygon, which is in turn added to $setOfPolygons$ in line 15. Finally, the generated polygons are returned (line 18).

Regarding the algorithm $generatePoints$ (Algorithm 2), it works as follows. In line 1, the variable $listOfPoints$, which represents the points to be returned, is initialized to empty and, in line 2, the

Algorithm 1: `BuildSquaredShapes` (setOfPoints)

```
   Input  : setOfPoints {set of points representing customers addresses}
   Output: setOfPolygons {set of squared shapes successfully created}
 1 count ← 0;
 2 d ← 0.003;
 3 setOfPolygons.New();
 4 while (count < setOfPoints.sizeOf()) do
 5     X_MIN ← (setOfPoints.getElement[count]).getX() - d;
 6     X_MAX ← (setOfPoints.getElement[count]).getX() + d;
 7     Y_MIN ← (setOfPoints.getElement[count]).getY() - d;
 8     Y_MAX ← (setOfPoints.getElement[count]).getY() + d;
 9     shift ← (X_MAX - X_MIN) / 100;
10     polygon.New();
11     polygon.AddPoints(generatePoints (X_MIN, Y_MIN, Y_MAX, shift, "Vertical"));
12     polygon.AddPoints(generatePoints (X_MAX, Y_MAX, Y_MIN, shift * (-1), "Vertical"));
13     polygon.AddPoints(generatePoints (Y_MIN, X_MAX, X_MIN, shift * (-1), "Horizontal"));
14     polygon.AddPoints(generatePoints (Y_MAX, X_MIN, X_MAX, shift, "Horizontal"));
15     setOfPolygons.Add(polygon);
16     count ← count + 1;
17 end
18 return setOfPolygons;
```

value of $fixCoor$ is assigned to the auxiliary variable $auxCoor$. The size of the side in which the points will be generated is calculated and stored in the variable $sizeOfSegment$ in line 3, while the variable $totalShift$, which represents the offset from the beginning of the side that will contain the point being generated, is initialized in line 4. Lines 6 to 10 create a point for a given side of the polygon. If this side corresponds to a vertical line segment, the point with coordinates ($auxCoor$, $varCoor + totalShift$) is created; otherwise the coordinate values are reversed and the point ($varCoor + totalShift$, $auxCoor$) is created. Then, the value of $totalShift$ is increased by $shift$ (line 11) aiming to determine the coordinate of the new point along the side, and the value of $auxCoor$ is updated in lines 13 and 15 to slightly change the shape of the polygon. The loop of lines 5 to 17 is performed until the maximum coordinate of a side be reached. Finally, the generated points are returned (line 18).

Algorithm 2: `generatePoints` ($fixCoor$, $varCoor$, $maxCoor$, $shift$, $typeOfLine$)

```
   Input  :fixCoor {fixed coordinate of the points generated along a side}
           varCoor {initial value of the coordinate of the points generated along a side}
           maxCoor {maximum coordinate of a side}
           shift {a small shift to generate polygons to slightly differ from a square}
           typeOfLine {horizontal or vertical side}
   Output: list of Points {list of points that compose the line segment}
 1 listOfPoints.New();
 2 auxCoor ← fixCoor;
 3 sizeOfSegment ← abs(maxCoor − varCoor);
 4 totalShift ← 0;
 5 while ((sizeOfSegment − abs(totalShift)) > 0) do
 6     if typeOfLine = "Vertical" then
 7         listOfPoints.Add(auxCoor, varCoor + totalShift);
 8     else
 9         listOfPoints.Add(varCoor + totalShift, auxCoor);
10     end
11     totalShift ← totalShift + shift;
12     if abs(totalShift) < sizeOfSegment/2 then
13         auxCoor ← auxCoor - 0.000001;
14     else
15         auxCoor ← auxCoor + 0.000001;
16     end
17 end
18 return listOfPoints;
```

4.2 Performance Evaluation

Aiming to investigate the impact of join operations costs, we conducted experiments that issued SOLAP queries related to the lowest granularity level of both the GHSSB and the NewGHSSB schemas.

We gathered the elapsed time in seconds to process these queries considering two different spatial data types on the attribute *c_ address_ geo*: point and polygon.

Our main performance findings are shown in the reduction column of Table VIII, which compares how much faster SOLAP queries were processed over the NewGHSSB schema than over the GHSSB schema. A positive value indicates that the NewGHSSB schema provided better performance results, while a negative value in this column indicates that the GHSSB schema was the best choice. The performance results showed that SOLAP queries had a great performance improvement over the NewGHSSB schema for point data. On the other hand, for polygon data, the performance was almost the same for both schemas, but slightly better over the GHSSB schema. In fact, queries related to point-based addresses (i.e. when addresses are represented as non-redundant points) caused a performance gain in the NewGHSSB schema of 27.82%, while queries related to polygon-based addresses (i.e. when addresses are represented as non-redundant polygons) resulted in a very small increase of 1.19% in the elapsed time over the NewGHSSB schema. Based on our experiments, we can conclude that the NewGHSSB schema should be chosen whenever non-redundant spatial data are represented as point objects, since this schema avoids additional joins between dimension and spatial dimension tables. On the other hand, if non-zero-dimensional geometries are used for representing non-redundant spatial data, the well-known GHSSB schema should be chosen.

Granularity Level	**Query Processing (elapsed time in seconds)**		**Reduction(%)**
	GHSSB	**NewGHSSB**	
ADDRESS (Point Data)	135.15	105.74	27.82
ADDRESS (Polygon Data)	110.74	112.08	−1.19

Table VIII. Test results for the non-redundant schemas.

5. GUIDELINES FOR BUILDING A LOGICAL GDW SCHEMA

From the performance evaluation discussed in Sections 3 and 4, we specified a taxonomy of tables and a set of guidelines for helping the design of spatial dimension tables of logical GDW schemas. These tables are detailed as follows, together with the proposed set of guidelines called "Logical GDW Design Guidelines".

Guideline 1: *Spatial dimension tables can have vector geometry data in any dimensional level and are represented using the following tables: Geometric, Descriptive and Descriptive-and-Geometric.*

We consider that spatial dimension tables are traditional dimensions with attributes that are used for directly storing vector geometries or for keeping references to these geometries (i.e. by indirectly storing spatial data). Table IX lists the types of tables that should be taken into account when creating spatial dimensions and building a logical GDW schema. These tables are classified according to whether they have been designed for keeping spatial data only, for storing conventional data only or for maintaining both vector geometry data and conventional attributes in a single table.

Guideline 2: *All redundant N-dimensional spatial data ($N \geq 0$) must be stored using two tables, namely Descriptive and Geometric, to avoid spatial data redundancy. Each redundant conventional data is stored in a table of type Descriptive together with a foreign key that points to its corresponding spatial data that are recorded non-redundantly in a table of type Geometric.*

The table *Geometric* listed in the first row of Table IX only stores a primary key and a spatial level whose members are non-redundant vector geometry data. The tables *City*, *Nation*, *Region*, *S_ Address* and *C_ Address* shown in Figure 2 are examples of this table type. Also, the table *Descriptive* has a primary key, a set of conventional levels or properties that are represented only by conventional data that can be either location descriptions (e.g. customers' address) or non-location descriptions (e.g. gender) and a set of foreign keys where each of these foreign keys refers to a primary key of a

Table	Objective	Column	Attribute Data Type
Geometric	To avoid the redundant storage of geometries shared among several spatial dimension tables.	Primary Key	Sequence Number
		Spatial Level	Non-redundant N-dimensional spatial data (N $\geq$ 1)
Descriptive	To store descriptive data used for categorizing and grouping facts and to represent relationships between the tables *Geometric* and *Descriptive-and-Geometric*.	Primary Key	Sequence Number
		Set of Conventional Levels	A Conventional Data Type of SQL per Level
		Set of Foreign Keys	A Sequence Number per Key
Descriptive-and-Geometric	To avoid join operations to access non-redundant point data and to avoid the storage of N-dimensional spatial data (N $\geq$ 1) together with their conventional attributes.	Primary Key	Sequence Number
		Set of Conventional Levels	A Conventional Data Type of SQL per Level
		Set of Spatial Levels	Non_redundant Point Data per Level
		Set of Foreign Keys	A Sequence Number per Key

Table IX. A taxonomy of tables for building logical GDW schemas.

Geometric table. The tables *Customer* and *Supplier* shown in Figure 1 are instances of a table of type *Descriptive*. For the sake of simplicity, we do not make a distinction between a location description and any other kind of conventional data. This is related to the fact that conventional data always require less storage space than spatial ones.

Guideline 3: *Non-redundant point data must be stored together with their conventional data using the Descriptive-and-Geometric table.*

The table *Descriptive-and-Geometric* contains a primary key, a set of conventional granularity levels or properties that are represented only by conventional data that can be either location descriptions or non-location descriptions, and a set of spatial granularity levels whose members of a given level are non-redundant *point* coordinate data and for the remaining levels, their members must be represented by foreign keys pointing to the corresponding primary keys of a *Geometric* table. The tables *Customer* and *Supplier* given in Figure 5 are examples of this table definition.

Guideline 4: *Spatial dimensions store redundant conventional data of spatial objects together with their corresponding spatial data in a single table called Descriptive-and-Geometric only if spatial data are non-redundant point data or N-dimensional spatial data (N $\geq$ 1) represented by a set of foreign*

keys.

Guideline 5: *Spatial dimensions store redundant conventional data of spatial objects separately from their corresponding spatial data in a pair of tables called Descriptive and Geometric, respectively, only if these geometries are redundant data or N-dimensional spatial data ($N \geq 1$).*

When building a logical GDW schema, if all of its spatial dimension tables are instances of any of the three table types listed in Table IX and are designed according to Guidelines 1 to 5, the resulting logical GDW schema is seen as non-redundant. This is due the fact that in this schema vector geometry data are not stored redundantly and are stored separately from their corresponding conventional data if they are non-zero-dimensional spatial objects. The GDW schemas shown in Figures 2 and 5 are examples of logical non-redundant schemas.

Finally, a logical GDW schema may also contain *non-spatial dimensions*, which stores only conventional data and do not include foreign keys to any other table with spatial data. The table *Date* and *Part* shown in Figure 2 are examples of this dimension type.

6. RELATED WORK

Several techniques have been proposed to improve SOLAP query processing performance over GDW. They can be classified according to the following groups: (i) the use of materialized views [Harinarayan et al. 1996; Golfarelli et al. 2004]; (ii) the horizontal or vertical data fragmentation in one site or several sites in a distributed environment [Golfarelli et al. 2000; Ciferri et al. 2007; Costa and Madeira 2004]; (iii) the partition of data across multiple processors to enable parallel processing [Datta et al. 1998; Furtado 2004]; (iv) the use of index structures [O'Neil and Graefe 1995; Sarawagi 1997; Jürgens and Lenz 1999; Papadias et al. 2001; Siqueira et al. 2009a; 2009b]; and (v) the design of efficient data schemas to reduce query response times and minimize data storage costs [Fidalgo et al. 2004; Fonseca et al. 2007; Times et al. 2008; da Silva et al. 2010]. This last group is the focus of the work described here and therefore, the main studies related to this group are surveyed as follows.

Detailing GeoDWFrame [Fidalgo et al. 2004; Fonseca et al. 2007; Times et al. 2008], its GDW metamodel is based on the relational package of CWM (Common Warehouse Metamodel) [OMG 2001] and SFS (Simple Features Specification) for SQL of OGC [OGC 1999] to facilitate its usage and extension in other studies. It defines how the concepts (e.g. numeric measures and conventional and spatial dimensions) of a GDW schema can be organized and related to each other. It also provides a set of stereotypes with pictograms that are meant to assist and guide the user designer in his GDW modeling activity. In GeoDWFrame, a dimension table can be specialized in three different dimensions: (i) conventional dimension, which stores only conventional data, as in a traditional DW; (ii) hybrid dimension, which deals with location descriptions and conventional data (e.g. client's address stored as string plus age and gender); in this case location descriptions are not spatial data; and (iii) geographic dimension, which is specialized in composite and primitive dimensions. The geographic composite dimensions only stores location descriptions (e.g. country names), while the geographic primitive dimensions maintain the spatial data (e.g. geometries of countries) related to the corresponding location descriptions kept in composite or hybrid dimensions. Therefore, GeoDWFrame proposes the storage of conventional and spatial attributes separately in GDW schemas. Differently from GeoDWFrame, we focus on the joint storage of spatial and conventional data in a single dimension table according to the uniqueness and complexity of the spatial data. Another difference refers to the fact that GeoDWFrame has not been based on experimental validation, which is the case of the "Logical GDW Design Guidelines" proposed in this paper.

An extensive experimental performance evaluation aimed at validating GeoDWFrame was conducted in [Siqueira et al. 2009b]. Due to the intrinsic redundancy of dimensional data and the high costs of storing the spatial attributes (compared to foreign keys to primitive dimensions), keeping conventional and spatial attributes together in a single dimension table has proven to have high storage costs and

low performance. As a result, the Spatial Bitmap index (SB-index) was proposed in [Siqueira et al. 2009a] for improving SOLAP query performance on a redundant GDW. However, the impact of the increase in the number of joins to answer a given query that may refer to one or more query windows has so far not been studied. In fact, in this paper we extend and enrich the discussion introduced in [Siqueira et al. 2009b] by investigating how increasing numbers of query windows affect SOLAP query performance over GDWs. Another differential of our work is a performance comparison using two different non-redundant schemas to verify if a 1:1 association between *Descriptive* and *Geometric* dimensions and the complexity of the spatial objects enables the joint storage of conventional and spatial attributes.

7. CONCLUSION AND FUTURE WORK

In this paper, we analyzed the effects of spatial data redundancy in spatial analytical query performance over different types of geographic data warehouse schemas. We focused on a redundant schema, a non-redundant schema with vector geometry data stored separately from their corresponding descriptive data, and another non-redundant schema in which the joint storage of vector geometries with their descriptive data depends on both the spatial data redundancy and the spatial data type. We issued against these schemas spatial analytical queries that referred to different numbers of spatial query windows, generating a need for performing different numbers of join operations.

Our main findings showed that: (i) the chosen spatial representation for storing non-redundant spatial data does affect query processing performance in geographic data warehouses; (ii) if non-redundant spatial data are represented as point data, an approach to avoid additional join costs by storing both point data and their descriptive data in a single table should be chosen; (iii) even for lower granularity levels, if the number of disjoint query windows of spatial on-line analytical roll-up queries increases, spatial data redundancy does impair the performance of these queries; and (iv) redundant geographic data warehouse schemas introduce a severe drawback, as some spatial on-line analytical roll-up queries cannot reuse previously fetched spatial data, impairing query performance.

This paper also introduced a set of guidelines for guiding the design of logical geographic data warehouse schemas. The motivation behind proposing these guidelines are two-fold. First, they are based on quantitative evaluation results. Second, these guidelines may be useful for helping in the proposal of a normalization theory for geographic data warehouses.

The development of a normalization theory is seen as a first suggestion of future work. Also, it would be interesting to explore the use of real datasets to further compare non-redundant schemas. The definition of a spatial on-line analytical algebra is another additional research. Finally, extra investigations should focus on how predefined hierarchies and how spatial on-line analytical query processing based on spatial measures may be affected by spatial data redundancy.

Acknowledgments. This work has been supported by the following Brazilian research agencies: FAPESP, FACEPE, CNPq, CAPES, INEP and FINEP. The third and fourth authors also thank the support of the Web-PIDE Project in the context of the Observatory of the Education of the Brazilian Government.

REFERENCES

Câmara, G., Casanova, M. A., Hemerly, A. S., aes, G. C. M., and Medeiros, C. M. B. *Anatomia de Sistemas de Informação Geográfica.* X Escola de Computação, Campinas, SP, Brasil, 1996.

Chaudhuri, S. and Dayal, U. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Rec.* 26 (1): 65–74, 1997.

Ciferri, C. D. A., Ciferri, R. R., Forlani, D. T., Traina, A. J. M., and de Souza, F. F. Horizontal Fragmentation as a Technique to Improve the Performance of Drill-down and Roll-up Queries. In *Proceedings of the 2007 ACM Symposium on Applied Computing.* Seoul, Korea, pp. 494–499, 2007.

Costa, M. and Madeira, H. Handling Big Dimensions in Distributed Data Warehouses Using the DWS Technique. In *Proceedings of the ACM 7th International Workshop on Data Warehousing and OLAP*. Washington, DC, USA, pp. 31–37, 2004.

da Silva, J., de Oliveira, A. G., Fidalgo, R. N., Salgado, A. C., and Times, V. C. Modelling and Querying Geographical Data Warehouses. *Inf. Syst.* 35 (5): 592–614, 2010.

Datta, A., Moon, B., and Thomas, H. A Case for Parallelism in Data Warehousing and OLAP. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*. Vienna, Austria, pp. 226–231, 1998.

Demers, M. N. *Fundamentals of Geographic Information Systems*. John Wiley & Sons, 2000.

Fidalgo, R. N., Times, V. C., da Silva, J., and de Souza, F. F. GeoDWFrame: A Framework for Guiding the Design of Geographical Dimensional Schemas. In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery*. Zaragoza, Spain, pp. 26–37, 2004.

Fonseca, R., Fidalgo, R. N., da Silva, J., and Times, V. C. Um Metamodelo para a Especificação de Data Warehouses Geográficos. In *Proceedings of the 22th Brazilian Symposium on Databases*. João Pessoa, Paraíba, Brazil, pp. 193–207, 2007.

Furtado, P. Experimental Evidence on Partitioning in Parallel Data Warehouses. In *Proceedings of the 7th International Workshop on Data Warehousing and OLAP*. Washington, DC, USA, pp. 23–30, 2004.

Gaede, V. and Günther, O. Multidimensional Access Methods. *ACM Comput. Surv.* 30 (2): 170–231, 1998.

Golfarelli, M., Maio, D., and Rizzi, S. Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery*. London, UK, pp. 11–23, 2000.

Golfarelli, M., Maniezzo, V., and Rizzi, S. Materialization of Fragmented Views in Multidimensional Databases. *Data Knowl. Eng.* 49 (3): 325–351, 2004.

Harinarayan, V., Rajaraman, A., and Ullman, J. D. Implementing Data Cubes Efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Montreal, Quebec, Canada, pp. 205–216, 1996.

Jürgens, M. and Lenz, H.-J. Tree Based Indexes vs. Bitmap Indexes: A Performance Study. In *Proceedings of the International Workshop on Design and Management of Data Warehouses*. Heidelberg, Germany, pp. 14–15, 1999.

Kimball, R. and Ross, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., 2002.

Malinowski, E. and Zimányi, E. Representing Spatiality in a Conceptual Multidimensional Model. In *Proceedings of the ACM International Workshop on Geographic Information Systems*. Washington, DC, USA, pp. 12–22, 2004.

Malinowski, E. and Zimányi, E. Requirements Specification and Conceptual Modeling for Spatial Data Warehouses. In *Proceedings of the Workshop on Reliability in Decentralized Distributed Systems*. Montpellier, France, pp. 1616–1625, 2006.

Malinowski, E. and Zimányi, E. *Advanced Data Warehouse Design: from Conventional to Spatial and Temporal Applications (Data-Centric Systems and Applications)*. Springer Publishing Company, Inc., 2008.

OGC. Open GIS Consortium Inc.: Simple Features Specification for SQL Revision 1.1, 1999.

OMG. Common Warehouse Metamodel (CWM) Specification 1,1, 2001.

O'Neil, P., O'Neil, E., Chen, X., and Revilak, S. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Transaction Processing Performance Council Technical Conference*. Lyon, France, pp. 237–252, 2009.

O'Neil, P. E. and Graefe, G. Multi-Table Joins Through Bitmapped Join Indices. *SIGMOD Record* 24 (3): 8–11, 1995.

Papadias, D., Kalnis, P., Zhang, J., and Tao, Y. Efficient OLAP Operations in Spatial Data Warehouses. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*. Redondo Beach, CA, USA, pp. 443–459, 2001.

Rivest, S., Bedard, Y., Proulx, M., Nadeau, M., Hubert, F., and Pastor, J. SOLAP Technology: Merging Business Intelligence with Geospatial Technology for Interactive Spatio-temporal Exploration and Analysis of Data. *ISPRS Journal of Photogrammetry and Remote Sensing* 60 (1): 17–33, 2005.

Sarawagi, S. Indexing OLAP Data. *IEEE Data Eng. Bull.* 20 (1): 36–43, 1997.

Siqueira, T. L. L., Ciferri, R. R., Times, V. C., and Ciferri, C. D. A. A Spatial Bitmap-based Index for Geographical Data Warehouses. In *Proceedings of the 24th ACM Symposium on Applied Computing*. Honolulu, Hawaii, USA, pp. 1336–1342, 2009a.

Siqueira, T. L. L., Ciferri, R. R., Times, V. C., and Ciferri, C. D. A. The Impact of Spatial Data Redundancy on SOLAP Query Performance. *Journal of the Brazilian Computer Society* 15 (2): 19–34, 2009b.

Times, V. C., Fidalgo, R. N., ao da Fonseca, R. L., da Silva, J., and Oliveira, A. G. A Metamodel for the Specification of Geographical Data. In *Annals of Information Systems: New Trends in Data Warehousing and Data Analysis*. Springer, Chapter 5, pp. 1–22, 2008.