Investigating the Effects of Spatial Data Redundancy in Query Performance over Geographical Data Warehouses[1]

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

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – São Carlos, SP – Brasil

[2]Centro de Informática – Universidade Federal de Pernambuco – Recife, PE – Brasil

[3]Departamento de Ciências de Computação – Universidade de São Paulo
São Carlos, SP – Brasil

{thiago_siqueira,ricardo}@dc.ufscar.br,
vct@cin.ufpe.br, cdac@icmc.usp.br

Abstract. *Geographical Data Warehouses (GDW) are one of the main technologies used in decision-making processes and spatial analysis. For these, several conceptual and logical data models have been proposed in the literature. However, little attention has been devoted to the study of how spatial data redundancy affects query performance over GDW. In this paper, we investigate this issue. Firstly, we compare redundant and non-redundant GDW schemas and conclude that redundancy is related to high performance losses. Further, we analyze the indexing issue, aiming at improving query performance on a redundant GDW. Comparisons among the SB-index approach, the star-join aided by R-tree and the star-join aided by GiST showed that SB-index significantly improves the elapsed time on query processing from 25% up to 95%.*

1. Introduction

Although Geographical Information System (GIS), Data Warehouse (DW) and On-Line Analytical Processing (OLAP) have different purposes, all of them converge in one aspect: decision-making support. Some authors have already proposed their integration, by making use of a Geographical Data Warehouse (GDW) to provide a means of carrying out spatial analyses together with agile and flexible multidimensional analytical queries over huge data volumes [Fidalgo et al. 2004; Malinowski and Zimányi 2004; Silva et al. 2008; Sampaio et al. 2006]. However, little attention has been devoted to the investigation of the following issue: **how does the spatial data redundancy affect query response time and storage requirements in a GDW**?

Similarly to a GIS, a GDW-based system manipulates geographical data with geometric and descriptive attributes and supports spatial analyses and ad-hoc query windows as well. Like a DW [Kimball and Ross 2007], a GDW is a subject-oriented,

[1] This work has been supported by the following Brazilian research agencies: CAPES, CNPq, FAPESP, FINEP and INEP. The first two authors also thank the support of the Web-PIDE Project in the context of the Observatory of the Education of the Brazilian Government.

integrated, time-variant and non-volatile dimensional database, that is often organized in a star schema with dimension and fact tables. A dimension table contains descriptive data and is typically designed in hierarchical structures in order to support different levels of aggregation. A fact table addresses numerical, additive and continuously valued measuring data, and maintains dimensions keys at their lower granularity level and one or more numeric measures as well. In fact, a GDW stores geographical data into one or more dimensions or into at least one measure [Stefanovic et al. 2000; Fidalgo et al. 2004; Malinowski and Zimányi 2004; Sampaio et al. 2006; Silva et al. 2008]. OLAP is the technology that provides strategic multidimensional queries over the DW. Further, Spatial OLAP (SOLAP) tools provide analytical multidimensional queries based on spatial predicates that mostly run over GDW [Gaede and Günther 1998; Kimball and Ross 2002; Fidalgo et al. 2004; Silva et al. 2008]. Often, the spatial query window is an ad-hoc area, not predefined in the spatial dimension tables.

Attribute hierarchies in dimension tables lead to an intrinsically redundant schema, which provides a means of executing roll-up and drill-down operations. These OLAP operations perform data aggregation and disaggregation, according to higher and lower granularity levels, respectively. Kimball and Ross (2002) stated that a redundant DW schema is preferable to a normalized one, since the former does not introduce new joins costs and the attributes assume conventional data types that require only a few bytes. On the other hand, in GDW it may not be feasible to estimate the storage requirements for a spatial object represented by a set of geometric data [Stefanovic et al. 2000]. Also, evaluating a spatial predicate is much more expensive than executing a conventional one [Gaede and Günther 1998].

Hence, choosing between a normalized and a redundant GDW schema may not lead to the same option as for conventional DW. Then, an experimental evaluation approach should aid GDW designers to make this choice. In this paper we investigate spatial data redundancy effects in query performance over GDW. We mainly analyze if a redundant schema aids SOLAP query processing in a GDW, as it does for OLAP and DW. We also address the indexing issue aiming at improving query processing on a redundant GDW.

This paper is organized as follows. Section 2 discusses related work, Section 3 describes the experimental setup, and Section 4 shows performance results for redundant and non-redundant GDW schemas, using database systems resources. Section 5 describes indices for DW and GDW, and the SB-index structure that we proposed in Siqueira et al. (2009). Section 6 details the experiments regarding the SB-index for the redundant and non-redundant GDW schemas. Section 7 concludes the paper.

2. Related Work

Stefanovic et al. (2000) were the first to propose a GDW framework, which addresses spatial dimensions and spatial measures. Particularly, in a spatial-to-spatial dimension table, all levels of an attribute hierarchy maintain geometric features representing spatial objects. However, the authors do not discuss the effects of such redundant schema, by just focusing on the selective materialization of spatial measures. Fidalgo et al. (2004) foresaw that spatial data redundancy might deteriorate GDW performance. Because of this, they proposed a framework for designing geographical dimensional schemas that strictly avoid spatial data redundancy. The authors also validated their

proposal adapting a DW schema for a GDW. Even so, this validation work does not compare if the normalized schema performs SOLAP queries better than redundant ones. Finally, although Sampaio et al. (2006) have proposed a logical multidimensional model, they do not discussed about spatial data redundancy, but reuse the mentioned spatial-to-spatial dimension tables. Therefore, as far as we know, none of the related work outlined in this section have experimentally investigated the effects of spatial data redundancy in GDW, nor examined if this issue really affects SOLAP queries performance. This experimental evaluation is the main objective of our work.

3. Experimental Setup

3.1. Workbench

Experiments were conducted on a computer with 2.8 GHz Pentium D processor, 2 GB of main memory, 7200 RPM SATA 320 GB hard disk, Linux CentOS 5.2, PostgreSQL 8.2.5 and PostGIS 1.3.3. We have chosen the PostgreSQL/PostGIS DBMS because it is a prominent open source database management system. We also adapted the Star Schema Benchmark (SSB) [O'Neil, P., O'Neil, E. and Chen 2007] in order to support GDW spatial analysis, once its spatial information is strictly textual and maintained into Supplier and Customer dimensions. SSB is derived from TPC-H (www.tpc.org/tpch). The changes we made preserved descriptive data and created an spatial hierarchy based on the previously defined conventional dimensions. Both Supplier and Customer dimension tables have the following hierarchy: region $\prec$ nation $\prec$ city $\prec$ address. While the domains of the attributes *s_address* and *c_address* are disjoint, suppliers and customers share city, nation and region locations as attributes.

The following two GDW schemas were developed in order to investigate how much SOLAP queries performance is affected by spatial data redundancy. According to Stefanovic et al. (2000), Supplier and Customer are spatial-to-spatial dimensions and may maintain geographical data as shown in Figure 1: attributes with suffix "*_geo*" store the geometric data of geographical features, and there is spatial data redundancy. For instance, a map for Brazil is stored in every row whose supplier is located in Brazil. However, as stated by Fidalgo et al. (2004), in a GDW, spatial data must not be redundant and should be shared whenever is possible. Thus, Figure 2 illustrates Customer and Supplier sharing city, nation and region locations, but not addresses.

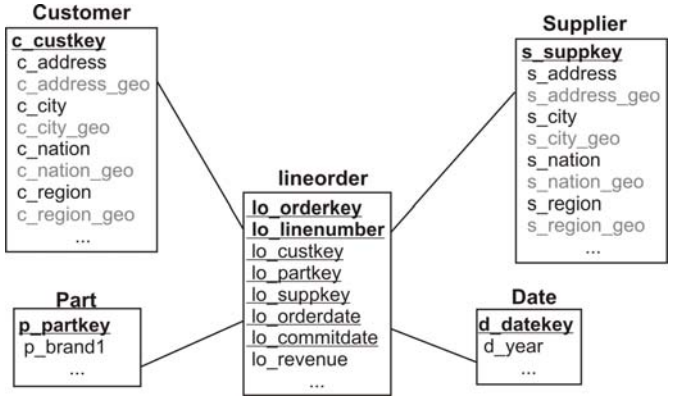

Figure 1. SSB adapted with spatial data redundancy: GRSSB.

While the schema shown in Figure 1 is named as GRSSB (Geographical Redundant SSB), the schema given in Figure 2 is called GHSSB (Geographical Hybrid SSB). Both schemas were created using SSB's database with scale factor 10. Data generation produced 60 million tuples in the fact table, 5 distinct regions, 25 nations per region, 10 cities per nation and a certain quantity of addresses per city that varies from 349 to 455. Cities, nations and regions were represented by polygons, while addresses were expressed by points. All geometries were adapted from Tiger/Line (www.census.gov/geo/www/tiger). GRSSB occupies 150 GB while GHSSB has 15 GB.

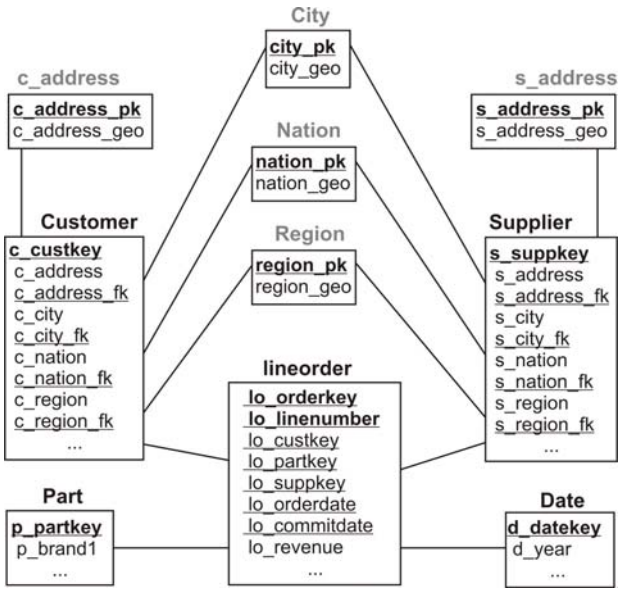

Figure 2. SSB adapted without spatial data redundancy: GHSSB.

3.2. Workload

The workload was based on Query Q2.3 from SSB, showed on Figure 3. We replaced the underlined conventional predicate with spatial predicates involving ad-hoc query windows (QW). These quadratic windows have had a correlated distribution with spatial data, and their centroids are random supplier addresses. Their sizes are proportional to the spatial granularity. Addresses were evaluated with containment range query [Gaede and Günther 1998] and its QW covers 0.001% of the extent. City, nation and region levels were evaluated with intersection range query [Gaede and Günther 1998] and their QW cover 0.05%, 0.1% and 1% of the extent, respectively.

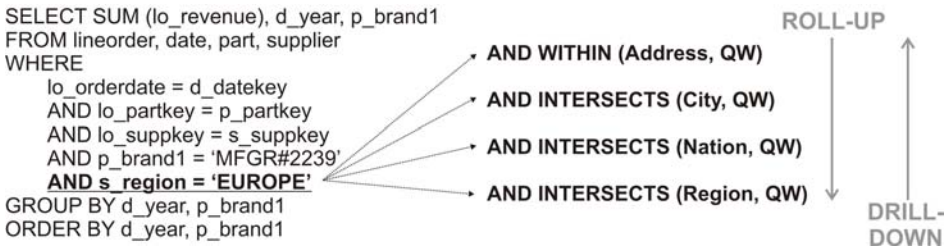

Figure 3. Adaption of SSB Q2.3 query.

The cited query windows allow the aggregation of data in different spatial granularity levels, i.e. the application of spatial roll-up and drill-down operations. Figure 3 also shows how complete spatial roll-up and drill-down operations are

performed. Both operations comprise four different size query windows that share the same centroid. Space limitations lead us to post further details about data and query distribution at http://gbd.dc.ufscar.br/papers/geoinfo08/figures.pdf.

4. Performance Results and Evaluation Applied for Different GDW Schemas

In this Section, we discuss the performance results related to the following two test configurations based on the GRSSB and GHSSB schemas: (C1) star-join computation aided by R-tree index [Guttman 1984] on spatial attributes; (C2) star-join computation aided by GiST index (http://gist.cs.berkeley.edu) on spatial attributes. The configurations reflect current available DBMS resources for GDW query processing. Experiments were performed by submitting 5 complete roll-up operations to GRSSB and GHSSB and by taking the average of the measurements for each granularity level. To analyze query processing we gathered the elapsed time (seconds). Table 1 shows these results.

Table 1. Performance results for configurations C1 and C2.

Query processing: elapsed time

	GRSSB C1	GHSSB C1	GRSSB C2	GHSSB C2
Address	2854.17	2866.51	2831.23	2853.85
City	2773.39	2763.17	2773.10	2758.70
Nation	4047.35	2766.14	3449.76	2765.61
Region	6220.68	2787.83	6200.44	2790.29

Fidalgo et al. (2004) foresaw that, although a normalized GDW schema introduces new join costs, it should require less storage space than a redundant one. Our quantitative experiments verified this fact and also showed that SOLAP query processing is negatively affected by spatial data redundancy in a GDW. According to our results, attribute granularity does not affect query processing in GHSSB. For instance, Region-level queries took 2787.83 seconds in C1, while Address-level queries took 2866.51 seconds for the same configuration, a very small increase of 2.82%. Therefore, our experiments showed that star-join is the most costly process in GHSSB. On the other hand, as the granularity level increases, the longer it takes to process a query in GRSSB. For example, in C2, a query on the finest granularity level (Address) took 2831.23 seconds, while on the highest granularity level (Region) it took 6200.44 seconds, an increase of 119%, despite the existence of efficient indices on spatial attributes. Therefore, it is possible to conclude that spatial data redundancy from GRSSB caused larger performance losses to SOLAP query processing, as compared to losses caused by additional joins in GHSSB.

Although our tests have pointed out that SOLAP queries processing in GHSSB is more efficient than in GRSSB, they also showed that both schemas offer prohibitive query response time. Clearly, the challenge in GDW is to retrieve data related to ad-hoc query windows and to avoid the high cost of star-joins. Thus, mechanisms to provide efficient query processing, as index structures, are essential. In the next sections, we describe existing indices and resume a novel index structure for GDW called SB-index [Siqueira et al. 2009], as well as discuss how spatial data redundancy affects them.

5. Indices

In the previous sections we identified some reasons for using an index structure to improve SOLAP query performance over GDW. Section 5.1 discusses the existing indices for DW and GDW, while in Section 5.2, we describe our SB-index approach.

5.1. Indices for DW and GDW

The Projection Index and the Bitmap Index were proven to be valuable choices in conventional DW indexing, since the multidimensionality is not an obstacle to them [O'Neil, P. and Quass 1997; Wu et al. 2006; Stockinger and Wu 2007; Wu et al. 2008]. Consider **X** as an attribute of a relation **R**. Then, the Projection Index on **X** is a sequence of values for **X** extracted from **R** and sorted by the row number. Surely, repeated values may exist. The basic Bitmap index associates one bit-vector to each distinct value ***v*** of the indexed attribute **X**. The bit-vectors maintain as many bits as the number of records found in the data set. If for the k-th record of the data set, we have that **X** = ***v***, then the k-th bit of the bit-vector associated with ***v*** has the value of one. The attribute cardinality, |**X**|, is the number of distinct values of **X** and determines the number of bit-vectors. Also, a star-join Bitmap index can be created for the attribute **C** of a dimension table, in order to indicate the set of rows in the fact table to be joined with a certain value of **C**.

Figure 4 shows data from a DW star-schema, a projection index and bit-vectors of two star-join Bitmap indices. The attribute s_suppkey is referenced by lo_suppkey shown in the fact table. Although s_address is not involved in the star join, it is possible to index this attribute by a star-join Bitmap index, since there is a 1:1 relationship between s_suppkey and s_address. $Q_1 \prec Q_2$ if, and only if it is possible to answer Q_1 using just the results of Q_2, and $Q_1 \neq Q_2$ [Harinarayan et al. 1996]. Therefore, it is possible to index s_region because s_region $\prec$ s_nation $\prec$ s_city $\prec$ s_address. For instance, executing a bitwise OR with the bit-vectors for s_address = B and s_address = C results in the bit-vector for s_region = 'EUROPE'. Visibly, creating star-join Bitmap indices to attribute hierarchies is enough to allow roll-up and drill-down operations.

DIMENSION TABLE: Supplier

s_suppkey	s_address	s_city	s_nation	s_region
1	A	VIETNAM 2	VIETNAM	ASIA
2	B	FRANCE 5	FRANCE	EUROPE
3	C	ROMANIA 2	ROMANIA	EUROPE
4	D	ALGERIA 6	ALGERIA	AFRICA
5	E	ALGERIA 0	ALGERIA	AFRICA

Projection Index

s_nation
VIETNAM
FRANCE
ROMANIA
ALGERIA
ALGERIA

FACT TABLE: Lineorder

lo_suppkey	lo_custkey	rev
1	235	20
1	512	16
2	512	22
3	235	19
3	512	15
3	106	21
4	235	20
5	106	18

Star-join Bitmap: s_address

A	B	C	D	E
1	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Star-join Bitmap: s_region

ASIA	EUROPE	AFRICA
1	0	0
1	0	0
0	1	0
0	1	0
0	1	0
0	1	0
0	0	1
0	0	1

Figure 4. Fragment of data, Projection and Bitmap indices.

Figure 4 also illustrates that instead of storing the value 'EUROPE' twice, the Bitmap Index associates this value to a bit-vector and uses 2 bits to indicate the 2 tuples where s_region = 'EUROPE'. This simple example shows how Bitmap deals with data redundancy. Moreover, if a query asking for s_region = 'EUROPE' and lo_custkey = 235 had been submitted for execution, then a bit-wise AND operation is executed using

the corresponding bit-vectors, in order to answer this query. Such property justifies why the number of dimension tables does not drastically affect the Bitmap efficiency.

High cardinality has been seen as the Bitmap's main drawback. However, recent studies have demonstrated that, even with very high cardinality, the Bitmap index approach can provide an acceptable response time and storage utilization [Wu et al. 2006, 2008]. Three techniques have been proposed to reduce the Bitmap index size [Stockinger and Wu 2007]: compression, binning and encoding. FastBit is a Free Software that efficiently implements Bitmap indices with encoding, binning and compression [Wu et al. 2006; O'Neil, E., O'Neil, P., and Wu 2007]. FastBit creates a Projection Index and an index file for each attribute. The index file stores one array of distinct values in ascending order, whose entries point to bit-vectors.

Although being suitable for conventional DW, to the best of our knowledge, the Bitmap approach has never been applied to GDW, nor FastBit has resources to support GDW indexing. In fact, just a single GDW index has so far been found in the database literature, namely the aR-Tree [Papadias et al. 2001]. The aR-tree index uses the R-tree's partitioning method to create **implicit ad-hoc hierarchies for the spatial objects**. Nevertheless, this is not the case for some GDW applications which mainly make use of predefined attribute hierarchies, such as region $\prec$ nation $\prec$ city $\prec$ address. Thus, there is a lack of an efficient index for GDW to support predefined spatial attribute hierarchies and to deal with multidimensionality.

5.2. The SB-index

The Spatial Bitmap Index (SB-index) aims at introducing the Bitmap index in GDW in order to reuse this method's legacy for DW and inherit all of its advantages. SB-index computes the spatial predicate and transforms it into a conventional one, which can be computed together with other conventional predicates. This strategy provides the whole query answer using a star-join Bitmap index. As a result, the GDW star-join operation is avoided and **predefined spatial attributes hierarchies can be indexed**. Therefore, the SB-index provides a means of processing SOLAP operations.

We have designed the SB-index as an adapted projection index. Each entry maintains a key value and a minimum bounding rectangle (MBR). The key value references the spatial dimension table's primary key and also identifies the spatial object represented by the corresponding MBR. DW often have surrogate keys, and thus, an integer value is an appropriate choice for primary keys. A MBR is implemented as two pairs of coordinates. As a result, the size of SB-index entry (denoted as **s**) is given by the expression **s** = sizeof(int) + 4* sizeof(double).

SB-index is an array stored on disk. L is the maximum number of SB-index objects that can be stored into one disk page. Suppose that the size **s** of an entry is equal to 36 bytes (one integer of 4 bytes and four double precision numbers of 8 bytes each) and that a disk page has 4 KB. Thus, we have L = (4096 DIV 36) = 113 entries. In order to avoid fragmented entries among different pages, some unused bytes (U) may separate them. In this example, U = 4096 – (113×36) = 28 bytes.

The SB-index query processing, illustrated in Figure 5, has been divided into two tasks. The first one performs a sequential scan on the SB-index and of adding

candidates (possible answers) to a collection. Then, the second task consists of checking which candidates are seen as answers and of producing a conventional predicate.

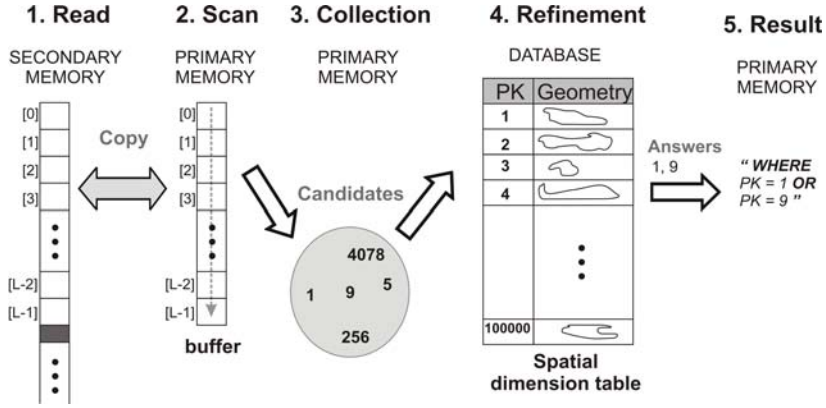

Figure 5. The SB-index Query Processing.

During the scan on the SB-index file, one disk page must be read per step, and its contents copied to a buffer in primary memory. After fulfilling the buffer, a sequential scan on it is needed to test, for each entry, the MBR against the spatial predicate. If this test evaluates to true, the corresponding key value must be collected. After scanning the whole file, a collection of candidates is found (i.e., key values whose spatial objects may be answers to the query spatial predicate). Currently, the SB-index supports intersection, containment and enclosure range queries, as well as point and exact queries.

The second task must check which candidates can really be seen as answers. This is the refinement task and requires an access to the spatial dimension table using each candidate key value, in order to fetch the original spatial object. Clearly, indicating candidates reduces the number of objects that must be queried, and is a good practice to decrease query response time. Then, the previously identified objects are tested against the spatial predicate using proper spatial database functions. If this test evaluates to true, the corresponding key value must be used to compose a conventional predicate. Here the SB-index transforms the spatial predicate into a conventional one. Finally, the resulting conventional predicate is passed to the FastBit software, which can combine it with other conventional predicates and resolve the entire query.

6. Performance Results and Evaluation Applied to Indices

In this Section, we discuss the performance results derived from the following additional test configuration: (C3) SB-index to avoid a star-join. This configuration was applied to the same data and queries used for configurations C1 and C2, and aims at comparing both C1 and C2 (given in Section 3) to C3. We reused the Experimental Setup outlined in Section 3. Our index structure was implemented with C++ and compiled with gcc 4.1. Disk page size was set to 4 KB. Bitmap indices were built with WAH compression [Wu et al. 2006] and without encoding or binning methods. To investigate the index building costs, we gathered: quantity of disk accesses, elapsed time (seconds) and space required to store the index. To analyze query processing we collected the elapsed time (seconds) and the number of disk accesses. In the

experiments, we submitted 5 complete roll-up operations to schemas GRSSB and GHSSB and took the average of the measurements for each granularity level.

6.1. First Battery of Tests

Table 2 shows the results derived from applying the SB-indices building operations to GRSSB and GHSSB. Due to spatial data redundancy, in GRSSB all indices have 100,000 entries (consequently the same size) and require the same number of disk accesses to be built. In addition, one spatial object has its MBR obtained several times, since this object may be found in different tuples, causing an overhead. The star-join Bitmap index occupies 2.3 GB, and took 11,237.70 seconds to be built. Each SB-index requires 3.5 MB, i.e., only a small addition to storage requirements (0.14% for each granularity level).

Table 2. Measurements on building SB-index for GRSSB and GHSSB.

	SB-indices building: GRSSB			*SB-indices building: GHSSB*		
	Elapsed time	**Disk accesses**	**Disk utilization**	**Elapsed time**	**Disk accesses**	**Disk utilization**
Address	48	886	3.5 MB	18	886	3.5 MB
City	1,856	886	3.5 MB	4	4	16 KB
Nation	11,566	886	3.5 MB	4	2	8 KB
Region	19,453	886	3.5 MB	2	2	8 KB

Due to normalization, in GHSSB, each spatial object is stored once in each spatial dimension table. Therefore, disk accesses, elapsed time and disk utilization assume lower values than that mentioned for GRSSB, during SB-index building. The star-join Bitmap index occupies 3.4 GB, and took 12,437.70 seconds to be built. SB-index adds at most 0.10% to storage requirements, specifically at the Address level.

Table 3 shows the SB-index query processing results. The time reduction columns compare how much faster C3 is than the best result between C1 and C2 (Table 1). SB-index can check if a point (address) is within a query window without refinement. However, to check if a polygon (city, nation or region) intersects the query window, a refinement task is necessary. Thus, Address level has a better performance gain.

Query processing using SB-index in GRSSB performed 886 disk accesses independently from the chosen attribute. This fixed number of disk accesses is due to the fact that we used sequential scan and all indices have the same number of entries for every granularity (i.e., all indices have the same size). Redundancy affected negatively SB-index, since the same MBR may be evaluated several times during both the SB-index sequential scan and the refinement task. On the other hand, adding only 0.14% to space requirements causes a query response time reduction between 25% and 95%, justifying the adoption of SB-index in GRSSB. In GHSSB, time reduction was always superior to 90%, even better than in GRSSB. This difference lies on the fact that indices had become smaller, due to spatial data redundancy avoidance. In addition, each MBR is evaluated only once, in contrast to GRSSB. Again, the tiny SB-index requires little storage space and contributes to reduce query response time drastically.

Table 3. Measurements on query processing with SB-index.

	Query processing: GRSSB			*Query processing: GHSSB*			
	C3 Elapsed Time	C3 Time reduction	C3 Disk accesses	C3 Elapsed Time	C3 Time reduction	C3 Disk accesses	
Address	132.41	95.32%	886	131.91	95.38%	886	**Address**
City	271.74	90.20%	886	150.00	94.56%	4	**City**
Nation	1178.12	65.84%	886	201.70	92.70%	2	**Nation**
Region	4621.99	25.45%	886	268.37	90.38%	2	**Region**

6.2 Redundancy Enhancement

As shown in Section 6.1, the motivation behind carrying out the simple and efficient SB-index modification is to be able to efficiently deal with spatial data redundancy. An overhead is caused while querying GRSSB because repeated MBR are evaluated and repeated spatial objects are checked in the refinement phase. On the other hand, unique MBR values should eliminate this overhead, as is the GHSSB case. Therefore, we proposed a second level to SB-index, which consists of assigning a list for each distinct MBR of all key values associated with it. The lists are stored on disk. On query processing, each distinct MBR must be tested against the spatial predicate. If this comparison evaluates to true, one key value from the corresponding list immediately guides the refinement. If the current spatial object is an answer, then all key values from the list are instantly added to the conventional predicate. Finally, the conventional predicate is passed to FastBit, which solves the entire query. As shown in Table 4, the modification severely decreased the number of disk accesses and the query response time in GRSSB. Address granularity level was not tested because it is not redundant. The adaption requires a very small fraction of the star-join Bitmap index volume: from 0.16% to 0.20% for City and Region granularity levels, respectively. Also, time reduction varies from 80.40% to 91.73%.

Table 4. Measurements on the enhancement of SB-index applied to GRSSB.

	Indices building			*Query processing*			
	Elapsed time	Disk accesses	Disk utilization	Disk accesses	Elapsed time	Time reduction	
City	2,005	101,385	4.81 MB	511	229.11	91.73%	**City**
Nation	11,428	100,935	3.93 MB	57	507.41	85.29%	**Nation**
Region	19,446	100,897	3.86 MB	36	1214.93	80.40%	**Region**

Finally, Figures 6 and 7 respectively shows how SB-index performed in GRSSB and GHSSB, by comparing it to the best result between C1 and C2. The axes indicate the time reduction provided by SB-index. In Figure 6, except for granularity level Address, the results are from SB-index second level enhancement. Instead of suggesting the elimination of the redundancy on GDW schemas, we proposed a means of reducing its effects. This strategy enhances SB-index portability, since it allows using the SB-index in distinct GDW schemas.

7. Conclusions

This paper has analyzed the effects of spatial data redundancy in query performance over Geographical Data Warehouses (GDW). In order to carry out such analysis, we compared the query response times of spatial roll-up operations for two distinct GDW

schemas. Since redundancy is related to attribute hierarchies in dimension tables, the first schema, GRSSB, was designed intrinsically redundant, while the other, GHSSB, avoids redundancy through normalization. Our performance tests, using current database management systems resources, showed that GRSSB's spatial data redundancy introduced more performance losses than GHSSB's joins costs.

These results motivated us to investigate indexing alternatives aiming at improving query performance on a redundant GDW. Comparisons among the SB-index, and the star-join aided by efficient spatial indices (R-trees and GiST) showed that SB-index greatly improved query processing: in GRSSB, performance gains were from 25.45% to 95.32%, while in GHSSB were from 90.38% to 95.38%. The results also showed that SB-index is very compact compared to the star-join Bitmap, requiring a very small fraction of this index volume: 0.10% to 0.14%. The lower improvement obtained in GRSSB motivated us to propose a specific enhancement on SB-index to deal with spatial data redundancy, by evaluating distinct MBR and spatial objects once, instead of multiple times. This enhancement provided performance gains from 80.40% to 91.73% in GRSSB. Clearly, spatial data redundancy is related to performance losses.

We are currently investigating some other strategies for minimizing the data redundancy effects on the SB-index, such as adapting a SB-index to use the R*-tree CR to manipulate distinct MBR [Beckmann et al. 1990; Gaede and Günther 1998]. In order to complement the investigation of the data redundancy effects, we are planning to run new experiments using increasing data volumes and different database management systems.

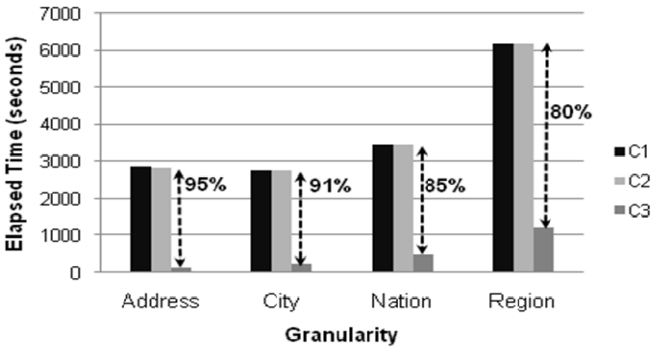

Figure 6. How SB-index performed better than C1 and C2 in GRSSB.

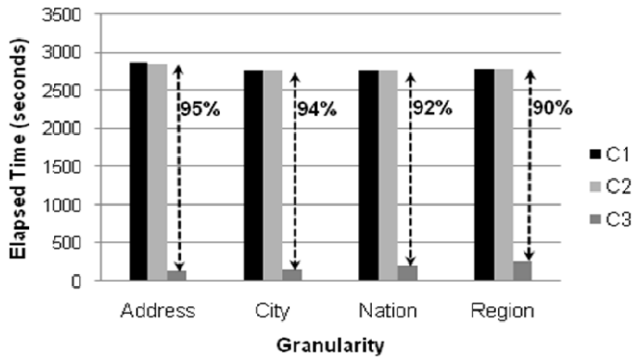

Figure 7. How SB-index performed better than C1 and C2 in GHSSB.

References

Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B. (1990) "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: SIGMOD Conference. p.322-331.

Fidalgo, R. N. et al. (2004) GeoDWFrame: "A Framework for Guiding the Design of Geographical Dimensional Schemas". In: 6th DaWak. p. 26-37.

Gaede, V., Günther, O. (1998) "Multidimensional Access Methods". In *ACM Computing Surveys*, v.30, n.2, p.170-231.

Guttman, A. (1984) "R-Trees: A Dynamic Index Structure for Spatial Searching". In *ACM SIGMOD Record*, v.14, n.2, p.47-57.

Harinarayan, V., Rajaraman, A. and Ullman, J. D. (1996) "Implementing Data Cubes Efficiently". In *ACM SIGMOD Record*, v.25, n.2, p.205-216.

Kimball, R. and Ross, M. (2002) The Data Warehouse Toolkit. Wiley, 2nd edition.

Malinowski, E. and Zimányi, E. (2004) "Representing Spatiality in a Conceptual Multidimensional Model". In: 12th ACM GIS. p.12-22.

O'Neil, E., O'Neil, P., and Wu, K. (2007) "Bitmap Index Design Choices and Their Performance Implications". In: 11th IEEE IDEAS. p. 72-84.

O'Neil, P., and Graefe, G. (1995) "Multi-Table Joins Through Bitmapped Join Indices". In *ACM SIGMOD Record*, v.24, n.3, p.8-11.

O'Neil, P., O'Neil, E. and Chen, X. (2007) "The Star Schema Benchmark", http://www.cs.umb.edu/~poneil/starschemab.pdf. July.

O'Neil, P. and Quass, D. (1997) Improved Query Performance with Variant Indexes, In: ACM SIGMOD Conference. p.38-49.

Papadias, D. et al. (2001) "Efficient OLAP Operations in Spatial Data Warehouses". In: 7th Symposium on Spatial and Temporal Databases. p. 443-459.

Sampaio, M. C., et al. (2006) "Towards a logical multidimensional model for spatial data warehousing and OLAP". In: 8th ACM DOLAP, p. 83-90.

Silva, J., Oliveira, A., Salgado, A. C., Times, V. C., Fidalgo, R., Souza C. (2008) "A set of aggregation functions for spatial measures". In 11th ACM DOLAP.

Siqueira, T. L. L., Ciferri, R. R., Times, V. C., Ciferri, C. D. A (2009) "An Spatial Bitmap-Based Index for Geographical Data Warehouses". In: 24th ACM SAC.

Stefanovic, N., et al. (2000) "Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes". In *IEEE TKDE*, v.12, n.6, p.938-958.

Stockinger, K. and Wu, K. (2007) "Bitmap Indices for Data Warehouses". In: Data Warehouses and OLAP: Concepts, Architectures and Solutions, Edited by Robert Wrembel and Christian Koncilia. IRM Press, p.157-178.

Wu, K., Otoo, E. J. and Shoshani, A. (2006) "Optimizing Bitmap Indices with Efficient Compression". In *ACM TODS* v.31, p.1-38.

Wu, K., Stockinger, K. and Shoshani, A. (2008) "Breaking the Curse of Cardinality on Bitmap Indexes". http://crd.lbl.gov/~kewu/ps/LBNL-173E.pdf, July.